1989

MacRootLocus, a CAD design tool for feedback control systems

Ko, Sung Hoon

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/26046

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MACROOTLOCUS, A CAD DESIGN TOOL
FOR FEEDBACK CONTROL SYSTEMS

by

Sung Hoon Ko

December 1989

Thesis Advisor: George J. Thaler

Approved for public release; distribution is unlimited

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b. OFFICE SYMBOL (If applicable) 62	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

MACROOTLOCUS, A CAD DESIGN TOOL FOR FEEDBACK CONTROL SYSTEMS

12. PERSONAL AUTHOR(S)

KO, Sung Hoon

13a TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	December 1989	

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	control systems; linear feedback systems, RootLocus Method; computer aided design; graphic analysis

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

MacRootLocus, a computer-aided design program, was developed as an analysis and design tool for linear feedback control systems. A variety of programs are presently available for the IBM-PC and IBM mainframe. The Apple Macintosh offers just a one-parameter root locus method capability. MacRootLocus supports both one-and two-parameter root locus methods on the Apple Macintosh. It is written in the computer language Turbo Pascal which is the native language of the Apple Macintosh and is designed with the same user friendliness and standard interface philosophy the Macintosh was designed for.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL George J. Thaler	22b TELEPHONE (Include Area Code) 408-646-2134; 22c OFFICE SYMBOL 62Tr

DD Form 1473, JUN 86 Previous editions are obsolete SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

UNCLASSIFIED

Appproved for Public Release; Distribution is Unlimited

MacRootLocus, A CAD Design Tool For Feedback Control Systems

by

Sung Hoon Ko
Major, Korean Air Force
B.S.E.E., Korean Air Force Academy, 1980

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

ABSTRACT

MacRootLocus, a computer–aided design program, was developed as an analysis and design tool for linear feedback control systems. A variety of programs are presently available for the IBM–PC and IBM mainframe. The Apple Macintosh offers just a one–parameter root locus method capability. MacRootLocus supports both one– and two–parameter root locus methods on the Apple Macintosh. It is written in the computer language Turbo Pascal which is the native language of the Apple Macintosh and is designed with the same user–friendliness and standard interface philosophy the Macintosh was designed for.

C. 1

TABLE OF CONTENTS

I. MACROOTLOCUS SYSTEM 1

A. INTRODUCTION 1

B. MATHEMATICAL CONCEPT 2

II. OPERATING THE MACROOTLOCUS 6

A. BASIC MACINTOSH USE 6

1. Basic Operation 6

2. Manipulating Window 8

3. Handling Input 9

4. Printing Out 11

B. MENUS AND DIALOGS 11

1. Apple Menu 12

2. File Menu 12

a. EQ Parameter 12

b. Get Coeff 13

c. Print Screen and Print Window 16

d. Quit 16

3. Edit Menu 16

4. Plot Menu 16

a. One Parameter 16

b. Two Parameter 19

5. Help Menu 20

6. Information box 21

III. DETAILED PROCEDURE MODULE DESCRIPTION 23
A. MAIN PROGRAM 23
1. Main Body 23
2. Handle Event Procedure 24
3. DoMouseDown Procedure 24
4. DoUpdate Procedure 25
5. DoKeyPress Procedure 25
6. HandleMenu Procedure 25
7. Initialization 26
B. GLOBALVAL UNIT 27
C. MAKEROOT UNIT 27
1. InfixPolish and ComputePolish Procedure 27
2. NextGain1 Function 32
3. NextGain2.Function 32
4. Results Procedure 32
5. PlotRootLocus1 and PlotRootLocus2 Procedure 32
D. ROOTFINDER UNIT 32
1. Laguerre's Method 33
2. InitAndTest Procedure 34
3. FindOneRoot Procedure 35
4. EvaluatePloy Procedure 35
5. ConstructDifference Procedure 35
6. TestForRoot Functions 35
7. ReducePoly Procedure 36

E. MESSAGE UNIT 36
1. SetUpWindow 36
2. MakeInfoScreen Procedure 36
F. MYDIALOG UNIT 37
G. TURBOGRAPH UNIT 38
IV. EXAMPLE OF DESIGN 40
A. OVER VIEW 40
B. GRAPHICAL SOLUTION 40
1. Example 1 (cascade lead compensation) 41
2. Example 2 (cascade lag compensation) 44
3. Example 3 (velocity feedback compensation) 46
4. Example 4 (velocity and acceleration feedback compensation) 49
V. CONCLUSION AND RECOMMENDATION 52
APPENDIX SOURCE CODE 53
LIST OF REFERENCES 181
INITIAL DISTRIBUTION LIST 182

I. MACROOTLOCUS SYSTEM

A. INTRODUCTION

MacRootLocus system is a program written for the Apple Macintosh computer. It was designed to allow a user to analyze and design linear feedback control systems. It is written in the computer language Turbo Pascal which is the native language of the Apple Macintosh and designed with the same user–friendliness and standard interface philosophy the Macintosh was designed for.

The Macintosh gets most of its input from the user through the 'mouse' which is a small cigarette pack–sized control that is moved across the table much like a pencil across paper to move the cursor or pointer on the computer screen. Rather than typing in commands like the IBM, the Macintosh lets you select the function you want performed with the mouse. Since the commands are not typed in, the commands do not have to be remembered as with other computers. It is the user–friendliness that sets MacRootLocus apart from other system analysis programs. Prior computer experience is not required to use MacRootLocus. A few minutes to learn how to use the mouse and pull down menus is all that is needed. All dialogs are easy to use and there are on–line Help menus. So the first–time user can get desired results without using trial and error.

Apple Macintosh models supported by MacRootLocus are the Macintosh SE and the Macintosh II system.

MacRootLocus was tested with several examples and is now available to any user on the Naval Postgraduate School Controls Laboratory computers under the icon of MacRootLocus system.

B. MATHEMATICAL CONCEPT

The root locus method is a graphical technique for determining the roots of the closed–loop characteristic equation of a system as a function of the static gain.

Consider the general feedback control system, as shown in Figure 1.1.

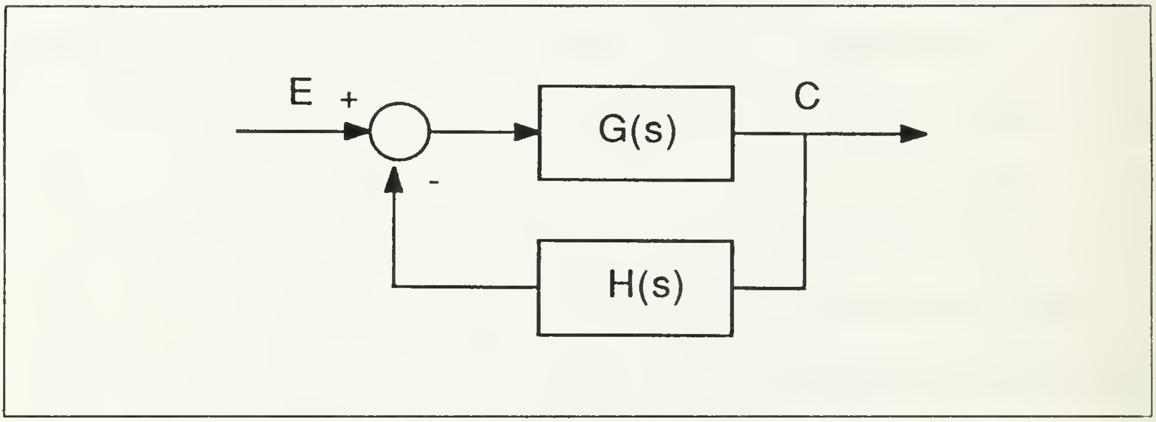

Figure 1.1 Closed–loop for Control System

In order to find the roots of the characteristic equation, it is required that

$$1 + G(s)H(s) = 0. \tag{1.1}$$

Of course, Eq. (1.1) may be rewritten as

$$G(s)H(s) = -1. \tag{1.2}$$

Since s is a complex variable, Eq. (1.2) may be rewritten in parametric form

$$|G(s)H(s)| = 1 \tag{1.3}$$

$$\angle G(s)H(s) = (2k-1)\pi \tag{1.4}$$

where k is an integer.

For a specific value of s to be a root of the characteristic equation, it must satisfy both Eqs. (1.3) and (1.4). Since the roots are those values of s that satisfy both equations, then the root points are points where these curves intersect.

The original development of the root locus method was concerned with the determination of the locus of roots of the characteristic equation as the system gain, K, is varied from zero to infinity. It appears that the root locus method is a single parameter method; fortunately it can be readily extended to the investigation of two or more parameters.

The characteristic equation of a dynamic system may be written as

$$a_n S^n + a_{n-1} S^{n-1} + \dots + a_1 S + a_0 = 0. \tag{1.5}$$

Clearly, the effect of the coefficient a_1 may be ascertained from the root locus equation

$$1 + \frac{a_1 S}{a_n S^n + a_{n-1} S^{n-1} + \dots + a_2 S^2 + a_0}. \tag{1.6}$$

The parameter of interest, A (in MacRootLous), can be isolated as

$$a_n S^n + a_{n-1} S^{n-1} + \dots + (a_{n-q} - A) S^{n-q} + A S^{n-q-1} + \dots$$
$$+ a_1 S + a_0 S = 0. \tag{1.7}$$

For example, a third–order equation of interest might be

$$S^3 + (3 + A)S^2 + 3S + 6 = 0. \tag{1.8}$$

In order to ascertain the effect of the parameter A, we isolate the parameter and rewrite the equation in root locus form as shown in the following steps:

$$S^3 + 3S^2 + AS^2 + 3S + 6 = 0 \tag{1.9}$$

$$1 + \frac{AS^2}{S^3 + 3S^2 + 3S + 6} = 0. \tag{1.10}$$

Then, to determine the effect of two parameters, we must repeat the root locus approach twice. Thus for a characteristic equation with two variable parameters, A and B (in Mac RootLocus),

$$a_n S^n + a_{n-1} S^{n-1} + \ldots . + (a_{n-q} - A)S^{n-q} + AS^{n-q-1} + \ldots$$

$$+ (a_{n-r} - B)S^{n-r} + BS^{n-r-1} + \ldots + a_1 S + a_0 = 0. \tag{1.11}$$

The two variable parameters have been isolated and the effect of A will be determined, followed by the determination of the effect of B. For example, for a certain third–order characteristic equation with A and B as parameters, we obtain

$$S^3 + S^2 + BS + A = 0. \tag{1.12}$$

In this particular case, the parameters appear as the coefficients of the characteristic equation. The effect of varying B from zero to infinity is determined from the root locus equation

$$1 + \frac{BS}{S^3 + S^2 + A} = 0. \tag{1.13}$$

One notes that the denominator of Eq (1.13) is the characteristic equation of the system with B = 0. Therefore, one first evaluates the effect of varying A from zero to infinity by utilizing the equation

$$S^3 + S^2 + A = 0 \tag{1.14}$$

rewritten as

$$1 + \frac{A}{S^2(S + 1)} = 0 \tag{1.15}$$

where B has been set equal to zero in Eq. (1.12). Then, upon evaluating the effect of A, a value of A is selected and used with Eq. (1.13) to evaluate the effect of B. This two–step method of evaluating the effect of A and then B may be carried out as a two–parameter root locus procedure. First, we obtain a locus of roots as A varies, and we select a suitable value of A; the results are satisfactory root locations. Then we obtain the root locus for B by noting that the poles of Eq. (1.13) are the roots evaluated by the root locus of Eq. (1.15).

In Mac RootLocus, this design approach is used to calculate the roots for the plot.

II. OPERATING THE MACROOTLOCUS

A. BASIC MACINTOSH USE

This document must serve as a user's manual as well as a technical explanation of the program and its capabilities. For this reason, the following brief explanation of Macintosh use is included. It is by no means a substitute for the Apple Macintosh Users' Manual but it will contain enough information for the beginner to be able to use Mac Root Locus.

1. Basic Operation.

The Macintosh has a finder, a special application you use to organize and manage your document and to start other applications. You use the finder every time you start your Macintosh, or whenever you move from one application to another.

The Macintosh screen looks like a light gray desktop, rather than a textual list of commands and responses. The desktop simulates the working environment. It is initially clean, displaying small graphic images, called icons, with short titles directly under them for each disk presently being used and a trash can in the lower right corner. An icon is an image representation of an application document or a control to a usable function. They offer quick recognition as to the type of item they describe and are easier to identify than lists or directories of file names with extensions.

The main interface between the user and Macintosh is the mouse. The Macintosh responds instantly to every movement you make with the mouse. You can start applications and get documents, work on them, and put them away again

just by moving the mouse and pressing the mouse button.

There are three techniques for the mouse: pointing, clicking and dragging. Moving the mouse moves the cursor or pointer on the screen in the same direction. Positioning the pointer on an item is called pointing to it. The mouse is used to select various items or icons on the screen. You select any item or icon to let Macintosh know this is what you want to work on next. You select icons by using a technique called clicking. As you click the icon, it becomes highlighted. This highlighting shows that you select it. This mouse can also be used to drag across something. This means the button is pressed and held while the mouse is moved. This action is called dragging. When dragging the mouse across the screen, a rectangle is outlined. When the button is released, everything within the rectangle is now highlighted and selected. Dragging also refers to moving items on the screen. This is done by pointing to an item, pressing and holding the button and moving the mouse. This will also move an outline of the icon selected and when the button is released, the icon is moved to the new location.

Whenever you work with Macintosh, you tell it two things: what you want to work on and what you want to do. First, you tell the Macintosh what you want to work on by selecting it as you have been doing with icons on the desktop. Then you tell the Macintosh what you want to do with the selection. You usually do this by choosing a command from a menu.

Along the top of the screen, in the menu bar, are titles of the menus. Pressing the mouse button while you are pointing to a menu title causes the title to be highlighted and a menu to appear, much like a window blind being pulled down. The menu contains commands you can carry out on what you have selected. Commands that you cannot use right now appear dimmed in the menu. When you

release the mouse button, the menu disappears.

To choose a command from a menu, you use the same dragging technique you used to move icons. As you drag through a menu, each usable command is highlighted in turn. If you change your mind about choosing a command, move the pointer off the menu and release the mouse button. Nothing is chosen unless you release the mouse button while one for the commands is highlighted. You'll follow this same pattern whenever you work with the Macintosh; 'select' some information, the 'choose' an action for it. For example in MacRootLocus, if you double click the mouse on MacRootLocus icon, the icon will be selected and MacRootLocus application will start running. You can see the menu bar along the top of the screen with the greeting message in the center of the screen. Now, you can select and choose the items to use MacRootLocus for your design.

2. Manipulating Window

The window is the area that displays information on the desk top. You view application documents and folders. Folders are a way of storing and organizing things, much like in a file drawer. Double clicking on a folder will open a window that displays its contents. Folders can be located in disks or within other folders. Usually 8 to 12 windows are the maximum that can be open at any one time. When several windows are displayed at once, they will usually overlap. If the window you want to view is partially covered by another window, pointing anywhere in your window and clicking will select it and make it the active window. The active window is always in front of all the others. It is the place you want the next action to happen, such as move or select icons or open other folders.

The active window can be identified by its highlighted title bar with narrow horizontal lines on either side of the title. The active window also usually

has a close box (to close the window) at the top–left corner; at the top–right corner is a zoom box that expands the window until it nearly covers the screen. On the bottom–right corner is the size box you use to change the size of a window. As you click or drag these boxes, you can get the function that you want.

3. Handling Input

Most information in the form of data or text is entered through the keyboard. The keyboard includes character keys, numerical keys, direction keys and other special keys. The return key tells the computer that the data just typed in should be accepted now. In MacRootLocus, the apple key in combination with another key is often a shortcut to choose a command from a menu. The tap key is used to move between data input points in the dialog for entering data. The direction key is important in MacRootLocus. It will be used for some word processing applications in a small box of the plot window. This will be further explained in the next section.

When the Macintosh requires information from you, it will display a dialog box like the one shown in Fig 2.1. It will tell you exactly what data is needed and shows you where to enter it. Data insertion points are small boxes in the dialog box that let you type in numbers or text. There are usually several such insertion points in each dialog box. The tab key lets you move from one point to another to enter data. Usually there will already be data in an insertion point box. This is the default data. You can change it if you want but you do not have to. When you enter new values in an insertion point box, they will become the new default values. Data insertion boxes can also be selected by pointing and clicking with the mouse. Clicking the mouse between two characters in the box will let you insert characters between them. Double clicking a box will highlight the entire number or an entire

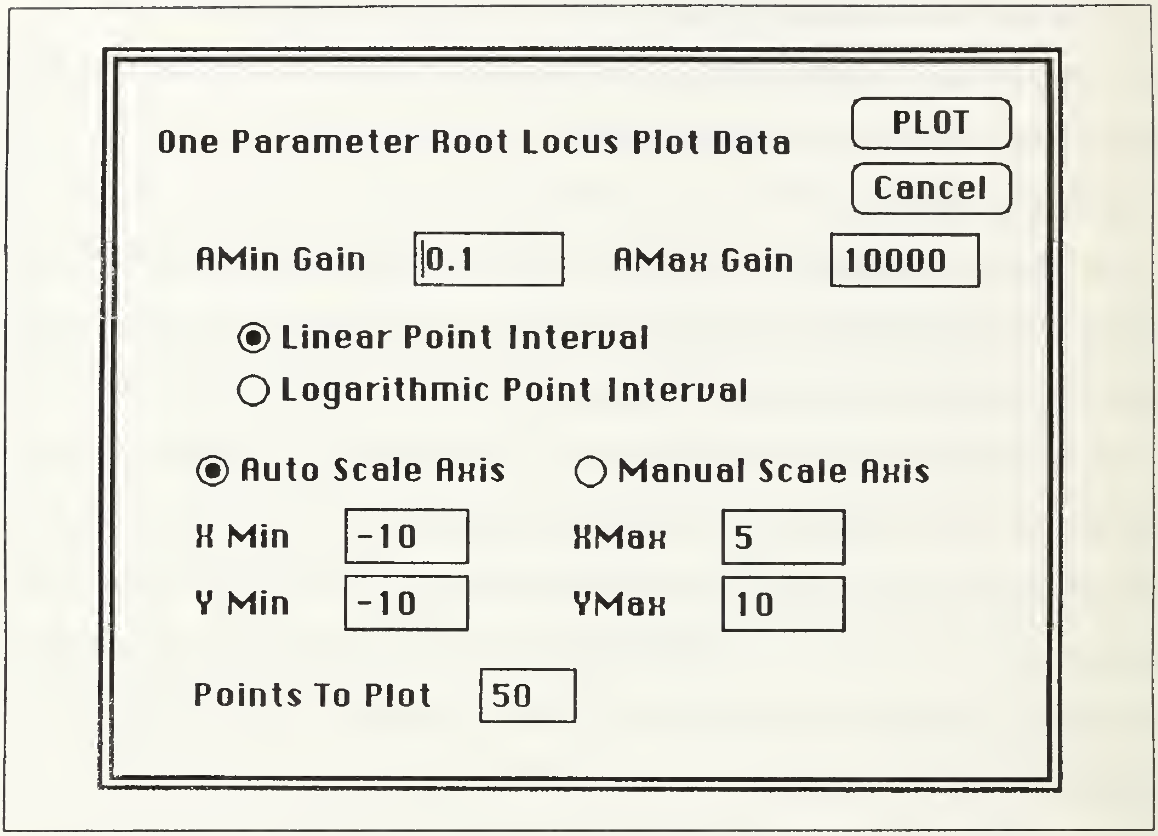

Figure 2.1 Sample Dialog

word if text is entered. You can also select all or portions of numbers or text by dragging the mouse across the text you want to select. When all or part of a text is selected, it will be highlighted. It can be removed by using the delete or backspace key, or it can be replaced by typing in whatever you want. After all data in the insertion box is correct, you can enter the data by hitting the return key or by clicking the OK button. If you click on Cancel, any changes to the data in the insertion boxes will not be saved and the operation which called the dialog box will be canceled. If you type in data that does not apply to the insertion box, such as

typing in letters in the AMin Gain insertion box of one–parameter plot data dialog, the Macintosh beeps. It informs you that you inserted the wrong input and you should correct the input that you just typed.

4. Printing Out

In order to print out your work, there are a few ways available in MacRootLocus. One way is to select the print command in the File menu. This allows the user to get a hard copy of any plot displayed by MacRootLocus.

Another way is to do a screen dump which will print the contents of the active window immediately. This is done by holding the command key and then typing the number '4'. This is a fast way to get print out of a plot in MacRootLocus. You can also create a MacPaint document by pressing the command and shift key and typing the number '3'. You can take up to 10 of these 'snapshots' and can then alter them with MacPaint for transferring to a word processor for lab writeup.

B. MENUS AND DIALOGS

There are five menus for MacRootLocus: Apple, File, Edit, Plot and Help. Actually, the File, Plot and Help menus are used to get the plot. These have to be used in order which will be explained later.

The Apple and Edit menu are not directly used by MacRootLocus, but they are used in order to follow the standard Macintosh programming philosophy for user–friendliness. This allows the experienced user to easily adapt to a new program since many of the operations are already familiar. Also, this is to allow for easy interaction with various other programs and desk accessories [Ref. 1].

As mentioned earlier, MacRootLocus presents commands in menus you pull

down from the menu bar. As soon as you choose the command you want, the dialog box appears for each command. This allows you to insert input data. Now more details for menus and dialogs in MacRootLocus will be explained.

1. Apple menu

The Apple menu is identified by a small apple in the top left corner. It is used primarily for desk accessories. It is also used by most programs as a way of offering program information. This is usually the first item of the Apple menu.

There are several accessories for some applications. For example, the Mac's clipboard and scrapbook are used to interact with the word processing program. It is highly recommended that the user read the Macintosh Users Guide as it explains the use of the Mac's Accessories which will be of use but will not be discussed here.

2. File Menu

MacRootLocus starts with the file menu first. It offers EQ Parameter, Get Coeff, Print Screen, Print Window and Quit commands. These selections also have a keyboard shortcut by holding down the command key, which has a clover leaf symbol on it, and hitting the first letter of the menu item at the same time.

a. EQ Parameter

The EQ Parameter stands for Equation Parameter. As mentioned the previous chapter, the MacRootLocus program calculates the roots of the characteristic equation for plotting points. It is neccessary to get the degree of polynomials and some equation parameters shown in Figure 2.3 since the Laguerre algorithm [Ref. 2] is used in order to calculate the roots.

The degree of the polynomial should be between 1 and 10 in MacRootLocus. Then there are the default values for the other parameters. These

values avoid the convergence error for almost all polynomials. But if convergence error messages appear on the screen, you can change these parameter values. These parameters must satisfy the following conditions:

(1) Initial guess ≥ 0

(2) Maximum Iteration ≥ 100

(3) Tolerance > 0

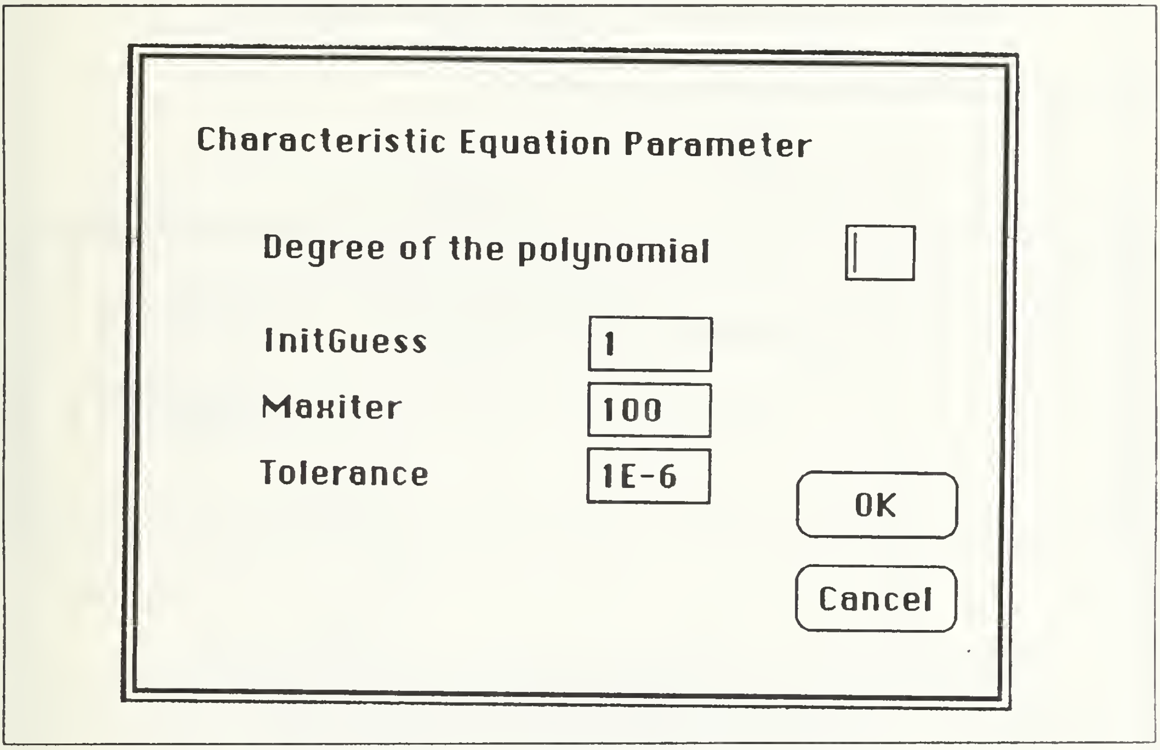

Figure 2.2 Equation Parameter Dialog

b. Get Coeff

After getting the degree of the polynomial, you should insert the coefficients of the polynomial. The 'Get Coeff' stands for Get Coefficients. The dialog for the coefficients is shown in Figure 2.3. This dialog is varied depending on

the degree of the polynomial as shown in Figures 2.3 and 2.4.

The algebraic expression for the coefficients of the characteristic equation of the system may have up to two undetermined parameters (A and B). In the case of the one–parameter root locus method, you use only one undetermined parameter (A). The routine uses standard algebraic, or infix, notation with parenthesis allowed. Operators can include +, –, *, /, and ˆ (exponentiation). The unary minus sign is allowed. For example, the characteristic equation is

$$S^3 + (10 + A)S^2 + (10 * A + 5000 * B)S + 5000 * A = 0$$

Figure 2.3 shows how you insert these coefficients.

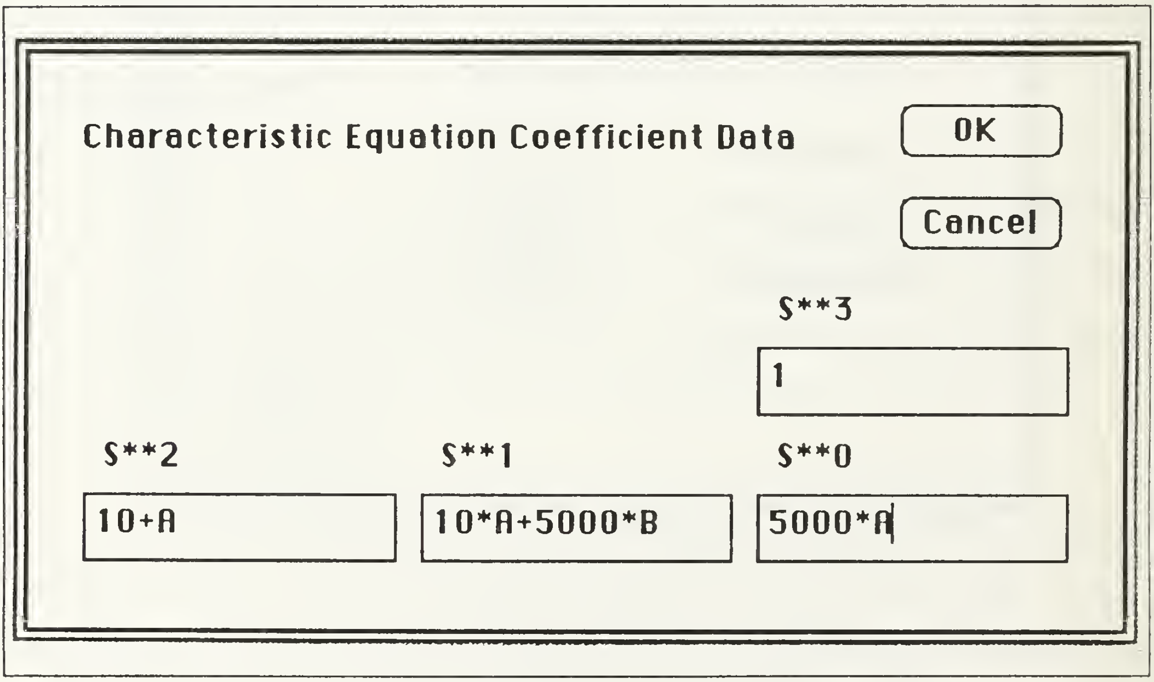

Figure 2.3 Characteristic Equation Coefficient Data Dialog Box for Third Degree Polynomial

If you choose the Get Coeff command without the degree of the polynomial,

the message shown in Figure 2.5 appears on the screen. This message tells you that degree of the polynomial has not yet been entered.

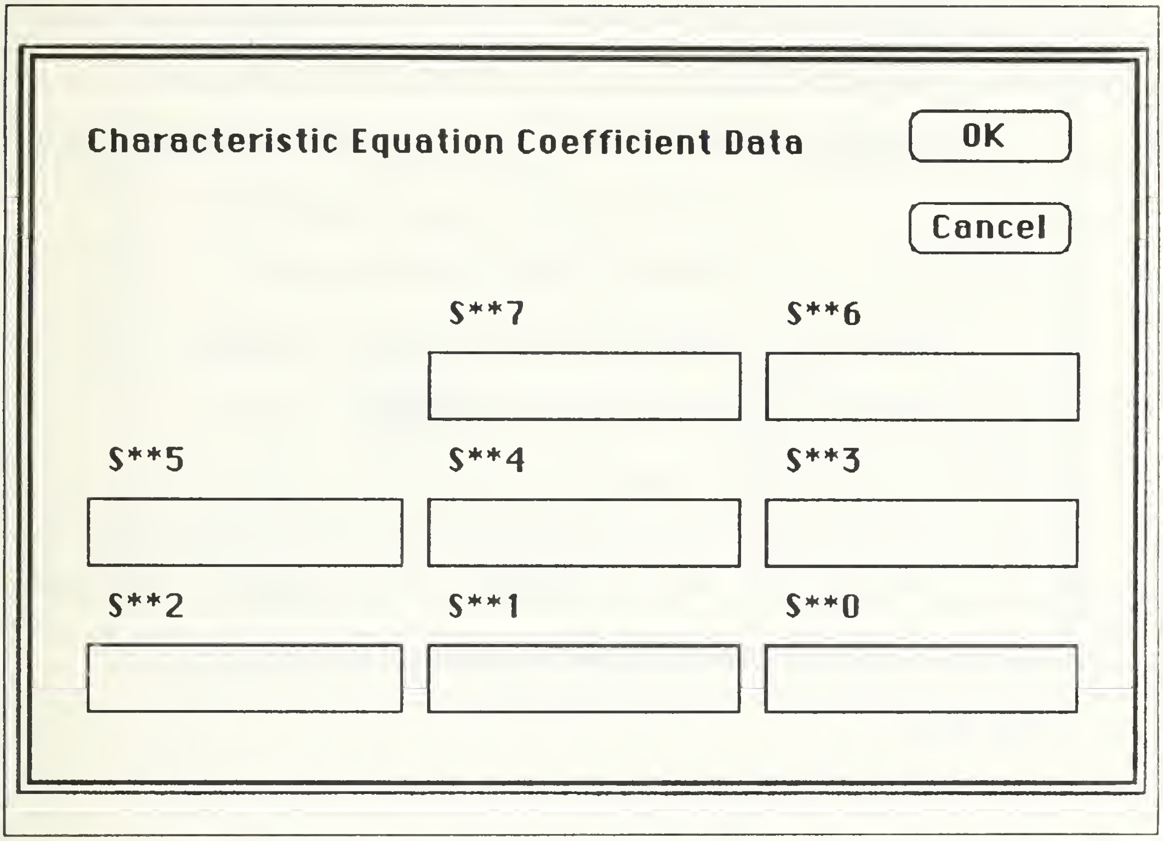

Figure 2.4 Characteristic Equation Coefficient Data Dialog Box for Seven Degree of Polynomial

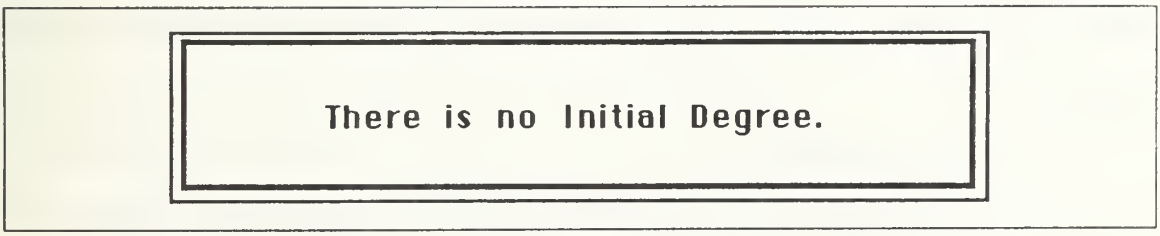

Figure 2.5 Message (1)

c. *Print Screen and Print Window*

These comands allow the user to get a hard copy of any plot displayed by MacRootLocus. When you choose the 'Print Screen' command, the whole contents of the screen is printed. If you choose the 'Print Window' command, the plot that is to be printed should be on the active window of the display.

Before attempting this method, ensure the printer is properly set up for friction feed.

d. *Quit*

The last item under the File menu is Quit. Selection of this will cause you to leave MacRootLocus and return to the desk top.

3. Edit Menu

Since these operations are not actually used in MacRootLocus, no further explanation of the Edit menu will be presented here. Any additional information regarding the Edit menu can be found in the Macintosh Users Manual.

4. Plot Menu

There are two commands, One Parameter and Two Parameter. These are called to display the dialog boxes of Figures 2.6 and 2.8 to insert the plot data.

These commands are followed after selecting the EQ Parameter and Get Coeff commands. If not, the message shown in Figure 2.7 appears to tell you that the degree of polynomial and polynomial coefficients should be entered before you choose the plot menu. After reading the message, just click once and this box will disappear.

a. *One Parameter*

When you choose the 'One Parameter' command, the dialog box shown in Figure 2.6 for plotting data of the one–parameter root locus method

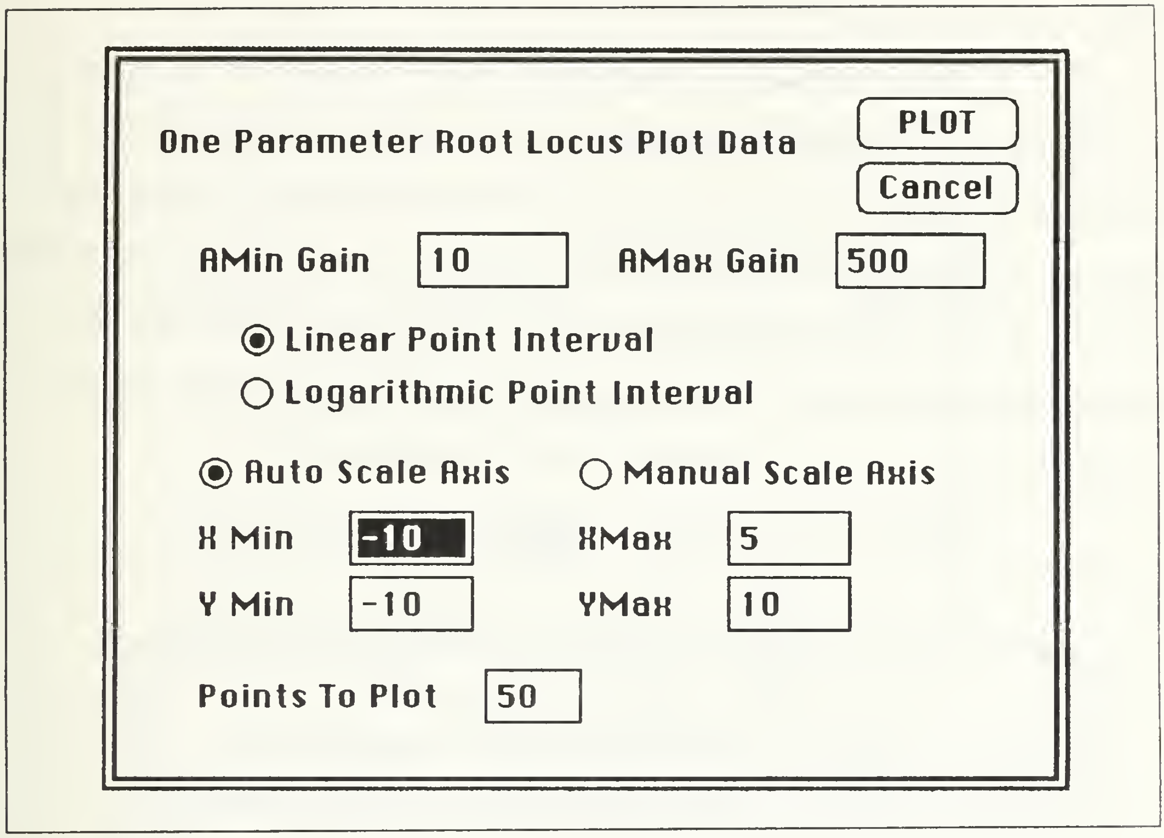

Figure 2.6 One Parameter Root Locus Plot Data Dialog

appears. The plot default values are shown in Figure 2.6. They can be changed as desired.

First, the user enters the minimum and the maximum gain values into the 'Min Gain' and the 'Max Gain' insertion box. Next, the user selects one of two types of interval, Linear and Logarithmic. When you click the radio button, the desired type of interval is chosen. For the 'Linear' interval, the gain step size is calculated by subtracting the minimum gain from the maximum gain entered in the dialog box, and then dividing by the number of points to plot. Using 'Logarithmic'

interval can best be described as giving equally spaced intervals on a logarithmic scale.

Most MacRootLocus programs calculate the gain intervals using the 'Linear' interval. This emphasizes gain values that are closer to the max gain. This becomes more evident as the maximum gain to the minimum gain ratio increases. Using the 'Logarithmic' interval gives more emphasis to the lower gains so more continuous loci can be drawn. As a basic rule of thumb, if the maximum gain to minimum gain ratio is greater than 100, selecting 'Logarithmic' interval will give a more continuous plot.

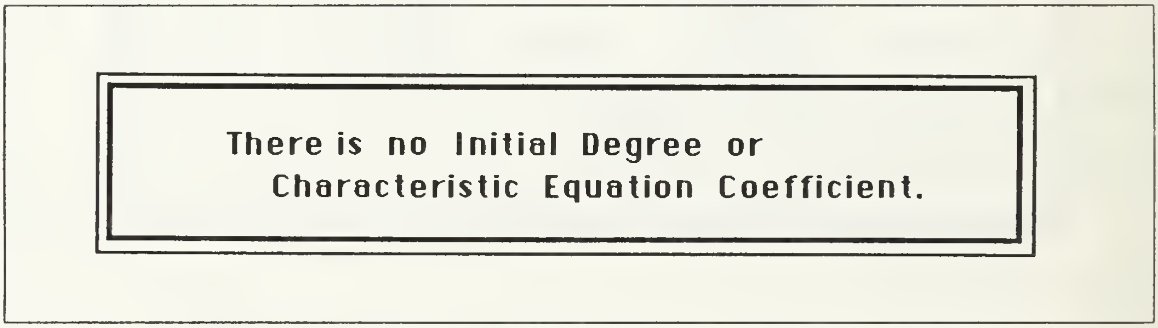

Figure 2.7 Message Box (2)

Next, the scale for the axis will be chosen. For the 'Auto Scale' the system calculates the minmum and maximum value of each axis. When the 'Manual Scale' is chosen, the user should insert the minimum and maximum values for each axis.

The last item, 'Points to Plot' sets the plot resolution. The bigger the value you choose, the better resolution plot you get, but the calculation time will be longer.

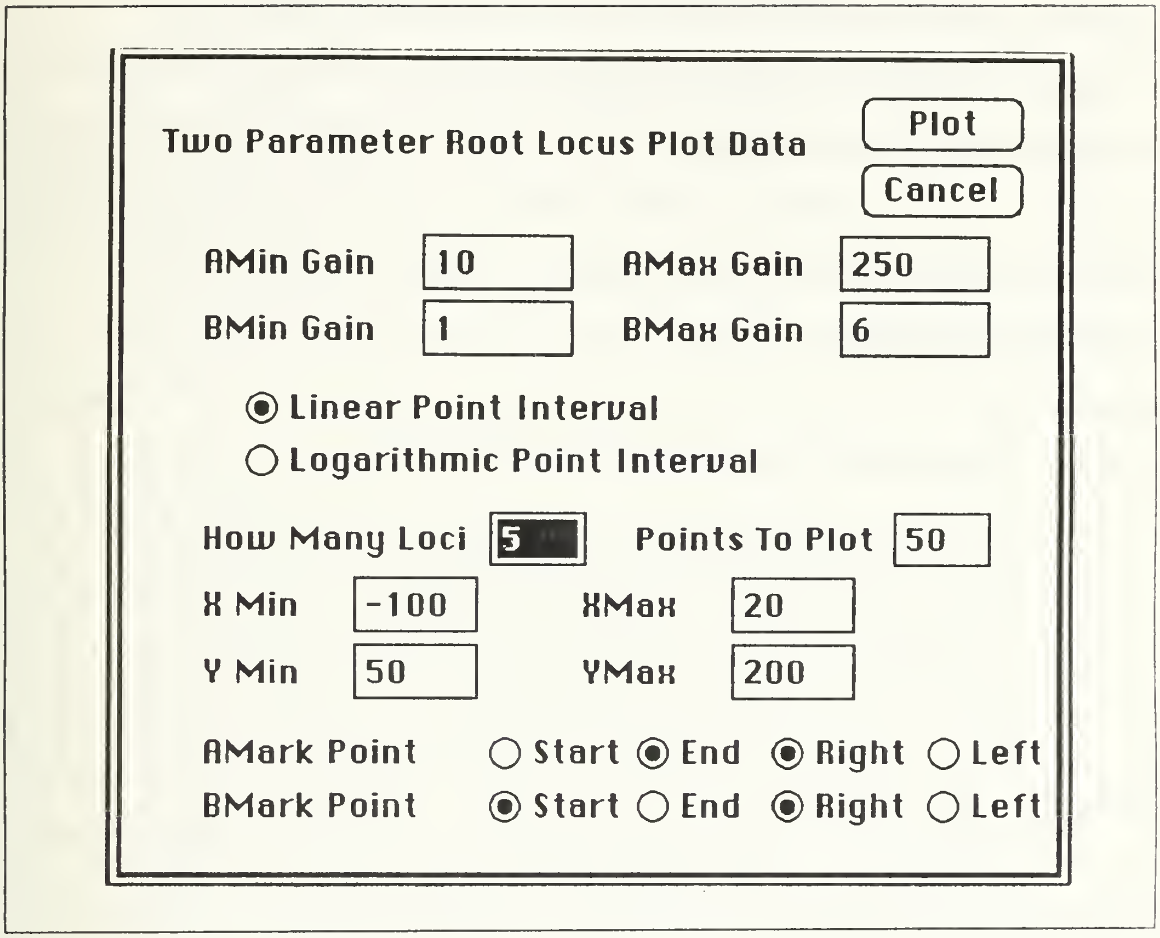

Figure 2.8 Two Parameter Root Locus Plot Data Dialog

b. Two Parameter

The 'Two Parameter' command calls the two–parameter plot data dialog. It is shown in Figure 2.8. The items shown in Figure 2.8 are similar to those shown in Figure 2.6, but several items are different.

There exists one more undetermined parameter 'B' to be inserted. The 'How many loci' item lets you decide how many loci are to be drawn for each parameter. The number of loci will be from 1 up to 10 for each parameter. In

Figure 2.8, this value is 5. There is no auto–scale for axes. Only the manual scale is available. You focus on the interesting area for your design. The last item is the marking and justification in order to draw the selected 'A' and 'B' values on the plot. There are four radio buttons. Two buttons are chosen each time for each parameter, one for position, the other justification to draw. There are sixteen combinations available for this work as shown in Table 2.1. Figure 2.10 shows you the ninth case in Table 2.1.

Table 2.1 The Combination for Marking and Justification

A Mark	Position	S	S	S	S	S	S	S	S	E	E	E	E	E	E	E	E
	Justification	R	R	R	R	L	L	L	L	R	R	R	R	L	L	L	L
B Mark	Position	S	S	E	E	S	S	E	E	S	S	E	E	S	S	E	E
	Justification	R	L	R	L	R	L	R	L	R	L	R	L	R	L	R	L

S : Start Point E : End point R : Right Hand Side L : Left Hand Side

5. **Help menu**

MacRootLocus supports an on–line help menu so that the first–time user can get desired results without using trial and error.

Help is the last item in the menu bar. There are the same item names in the menu bar. It makes it easy to look for the item for which the user wants information. The contents of each item are the subject of section 2.B.

6. Information Box

Finally, MacRootLocus gives you a convenient way to identify your plot. It is a small box in which you can type the information you want to memorize. As soon as the plot has been completed, the box appears at the bottom of the plot window automatically.

When the up direction key is pressed twice, the cursor comes out on the box. Then you can use the keyboard just like a typewriter. The capacity of the box is 160 to 200 letters. If you do not want that box, just click once. It will disappear. It is shown in Figure 2.10.

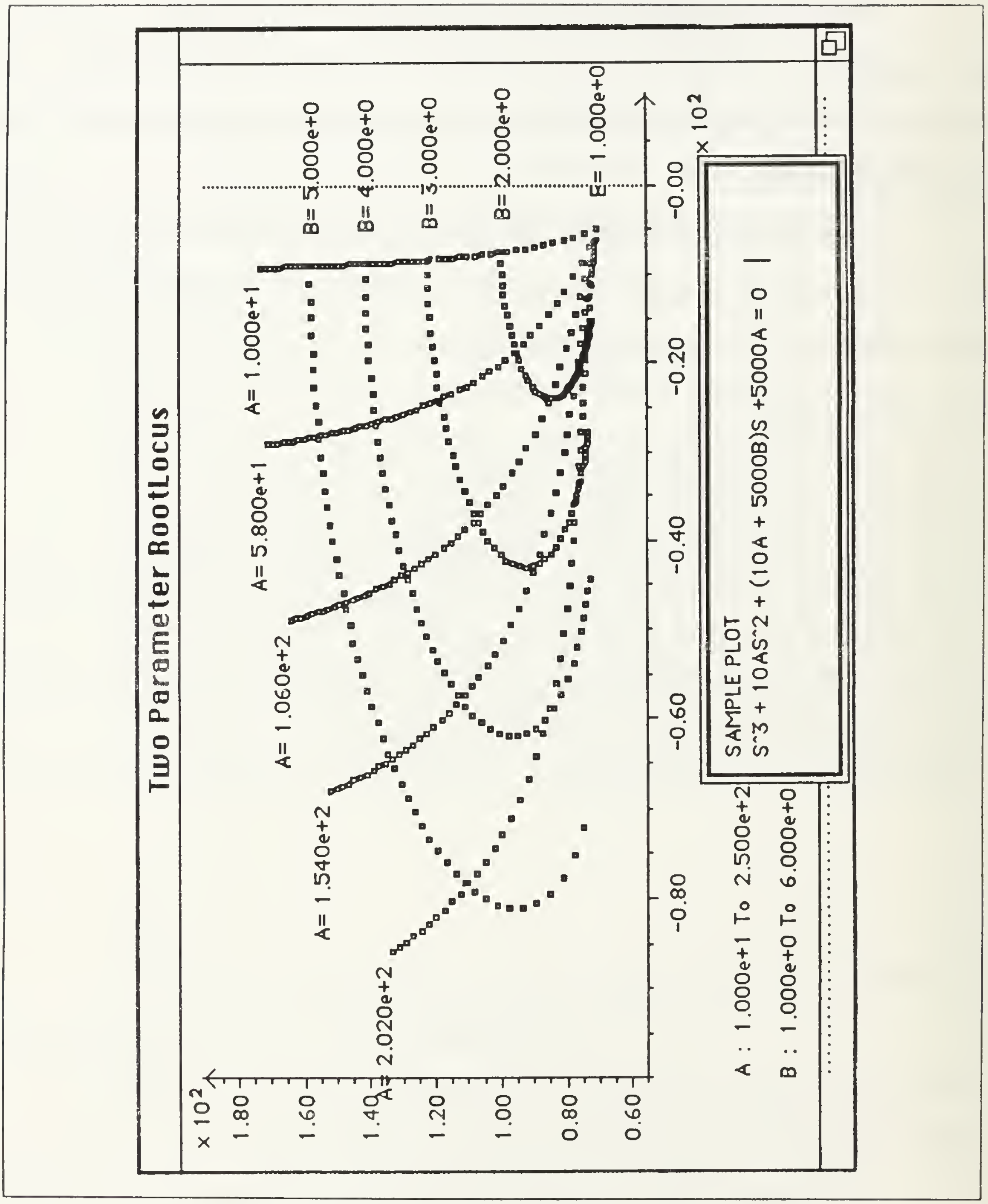

Figure 2.10 Sample Plot (1)

III. DETAILED PROCEDURE MODULE DESCRIPTION

This chapter will basically be programmer's notes covering the significiant procedures, functions and libraries to be used in MacRootLocus. MacRootLocus follows the Macintosh programming technique to make the program user friendly. It is a fairly simple Macintosh application that uses menus, windows, dialog boxes, and graphics. Most procedures and functions called in MacRootLocus were developed in Turbo Pascal verson 1.0 for Macintosh.

MacRootLocus consists of one main program, one resource file and six units. The main program integrates the resource file and units, then it shows you the menu bar and the greeting message to start the work. The details for these will now be explained.

A. MAIN PROGRAM

The Main Program is named MacRootLocus.Pas. This program consists of a main body and some precedures to handle the system.

1. Main Body

The structure of the main body uses the concept of event driven programming. It looks something like this:

```
Initialize;                                   { set everything up          }
repeat                                        { keep doing the following   }
  SystemTask;                                 { update desk accessories    }
  if GetNextEvent(everyEvent,theEvent) then   {   if there's an event...   }
    HandleEvent(theEvent);                    {     ...then handle it      }
until Finished;                               { until user is done         }
CleanUp;
```

The program is set up once with the user–defined routine 'Initialize.' It then enters a loop that continues until some condition (such as the user selecting Quit in a menu) causes it to set the boolean flag 'Finished' to true. Within that loop, it performs two major tasks.

First, it calls 'System Task' (a Toolbox routine), which allows the Mac operating system to update any desk accessories that might be in use. Second, it calls 'GetNextEvent' (another Toolbox routine) to see if any events have occurred. If any have, the highest priority event is returned in the data structure 'the Event.' The program then passes the event to 'Handle Event,' which is a user–defined routine that handles all the different events that might occur. Such events include key–presses, selection of menu items, mouse clicks, and windows being opened, closed, or resized. When the program is ready to terminate, it calls the user–defined routine clean up.

2. Handle Event Procedure

When an event occurs, the operating system creates an event record and puts it in a queue, ready for you to handle. To see if there is one waiting, you call 'GetNextEvent,' a boolean function that returns true if there is an event there for you. You give it a mask of the events you are interested in; you can use the predefined mask 'Every Event' to look at all events. This event is passed to 'Handle Event,' which takes care of it . 'HandleEvent' is just a case statement using the 'what' field in 'the Event' to determine which of the procedures to call.

3. DoMouseDown Procedure

The routine 'DoMouseDown' determines which window the mouse was in when the clicking took place and where exactly it happened. Like 'HandleEvent,' 'DoMouseDown' is mostly a case statement.

4. DoUpdate Procedure

The Macintosh keeps track of a lot of things for you. For one, it tells you when some portion of a window needs to be drawn, because of resizing or removing a covering window. This is known as an update event, and it requires special handling. To handle an update event, the routine 'DoUpdate' saves the current 'grafport' into 'SavePort' and makes 'the Window' the current port so that you can write to it. 'BeginUpdate' limits all output to the section of 'theWindow' that needs updating. You then do whatever redrawing is needed. When you are done, 'EndUpdate' lifts those limits, and 'SetPort(SavePort)' restores the old 'grafport'.

5. Dokeypress Procedure

This routine handles the 'Key down' and the 'Autokey' events. It is a check to see if a command–key combination was pressed; if so, it checks if the key is a menu command and takes appropriate action.

6. Handle Menu

The procedure 'Handle Menu' decodes the mouse position and figures out which menu and which item in that menu were selected. It uses a case statement to select the action for the appropriate menu; the menu value is the ID assigned when the menu is created. The commands in a menu are numbered from the top down, with the first command having a value of one. The action itself is usually a second case statement, based on the menu item.

When an item in a menu is selected, the name of that menu (in the menu bar at the top of the screen) is highlighted, that is, inverted to white–on–black. When you are done processing the menu command, the menu bar is restored to normal by calling 'HiliteMenu(0),' which is at the bottom of 'HandleMenu.' Another procedure is called before you can leave 'HandleMenu,' 'UpdateMenu,' a

local procedure that tests to see if certain items are to be enabled or disabled. To enable and disable menu items, the standard Macintosh procedures 'EnableItem' and 'DisableItem' are called.

7. Initialization

The initialization procedure for the MacRootLocus program includes the following structure: Call 'Init' routines, set up menus, set up windows, do other graphic initialization and do program–specific initialization.

There is an 'Init' routine for most of the major managers. The first, and most important, is 'InitGraf' (thePort). That sets up 'QuickDraw' (which is used by just about everything else) and sets up a 'grafport' for the screen. Other 'Init' routines are: InitFonts, InitWindow, InitMenus, TEInit, and InitDialogs (NIL).

Setting up menus involves four steps. First, it defines the menus themselves. If a resource file is used, just do a call to 'GetMenu' for each menu handle, or even a single call to 'GetNewMBar.' Otherwise, it has to build each menu using an initial call to 'NewMenu', followed by a call or calls to Append Menu. Second, if it is handling desk accessories, call 'AddResMenu'. Third, add all the menus to the menu bar by marking successive calls to 'InsertMenu.' 'Finally, call 'DrawMenuBar' to display the menu titles and make them active.

As with menus, the window initialization takes several steps. If MacRootLocus needs a window at start up, create it using either'GetNewwindow' (reading in from resources) or Newwindow (building it in place). Having created the window, make it the current 'grafport' by calling 'SetPort', then make it the active window by calling 'SelectWindow.'

The program–specific initialization should probably come here. All of

the default values for the dialogs are defined and the array vectors are initialized here.

B. GLOBALVAR UNIT

This unit declares all of the whole variables to be used in MacRootLocus except a few of local variables. This unit is called by the main program and all units. It defines constant, data type, and variables.

C. MAKEROOT UNIT

The unit MakeRoot is very important in MacRootLocus. This unit provides several procedures and functions to find the roots and the array vectors for plot. Since the characteristic equation is derived, the program must be able to parse the user's characteristic polynomial coefficient equations in order to understand the relations and be able to iteratively substitute in values for undetermined parameters: A (for one–parameter root locus method) or A and B (for two–parameter root locus method).

This unit has two main procedures, the Get Root 1 procedure for the one–parameter root locus method and the Get Root 2 procedure for two–parameter root locus method. The simplified algorithm for the two–parameter root locus method is outlined in Figure 3.1.

There are several procedures to perform this algorithm. These will be explained in the following section except for the Rootfinder unit. The InfixtoPolish and the ComputePolish are especially interesting.

1. InfixtoPolish and ComputePolish Procedure

The coefficients of the polynomial equation are written in algebraic, or

Given a system's characteristic polynomial :

$$C.E = a_n S^n + a_{n-1} S^{n-1} + \ldots + a_1 S + a_0 = 0$$

where a_n, a_{n-1}, etc. are algebraic expressions in A and B. Also A is to be stepped from the minimum A to the maximum A and B varied from the minimum B to the maximum B and the reverse is processed.

Then

```
SET A = Min A .  SET B = Min B.
SET A DeltaStep = abs((A Max - A Min) / Step)
SET B DeltaStep = abs((B Max - B Min) / Step)
  FOR 1 = 1 to 2 do      { case of A and B parameter}
    FOR j = 1 to Step do     {Step = quantity of losi}
      IF Step A then A = A Min + A DeltaStep * (j - 1)
      ELSE B = B Min + B DeltaStep * (j - 1)
      WHILE point no <= (points - 1) do
        IF Step A then B = NextGain2
        ELSE A = NextGain2
        FOR term = Initial Degree downto 0 do
          CONVERT ai from Infix to Polish
          SUBSTITUTE values for A and B
          COMPUTE ai
        END {FOR}
        CALL RootFinder
        CALL Results {make plot array and numerical data}
      END{WHILE}
      CALL PlotRootLOcus2
    END{FOR}
    Step A = False {Next case}
  END{FOR}
```

Figure 3.1 Two Parameter Root Locus Algorithm

'infix' notation. The available operators include (+) addition, (−) subtraction, (*) multiplication, (/) division, and (^) exponentiation. These operators follow a hierarchical precedence with exponentiation operations being done first, followed by multiplication and division, and finally addition and subtraction. Operations like

multiplication and division which have the same precedence are performed from left to right when conflicts arise. To change the order of precedence, parentheses may be used around any set of operations. These parenthetical expressions have the highest priority and, when nested, the innermost operations within parentheses are done first. This scheme follows closely the protocol used in most calculators and high level programming languages.

Infix notation, while convenient for the program user, does not lend itself well to computer manipulation. A better way to represent equations for the computer is the so called 'reverse Polish notation.' In reverse Polish notation, the operands of an equation are entered first, followed by the operator. For example, the infix expression

$$3 * 4 + 5$$

would be represented as

$$3\ 4 * 5 +$$

in reverse Polish notation. The numbers 3 and 4 are entered and multiplied, then 5 is entered and added to the previous result. Using the concept of a 'stack' the reverse Polish expression is easy to evaluate.

Recall that a stack is a last–in–first–out queue whose operation is analogous to a stack of trays. To operate the stack, the program calls a 'push' procedure to place an item on the stack, and a 'pop' procedure to remove the top item. Now, using the example given above, an arithmetic evaluation procedure can be illustrated. Figure 3.2 demonstrates such an implementation.

The basic equation evaluation algorithm can be outlined in three steps:

(1) Scan the reverse Polish equation term–by–term.

(2) If the term is a constant then push it onto the stack.

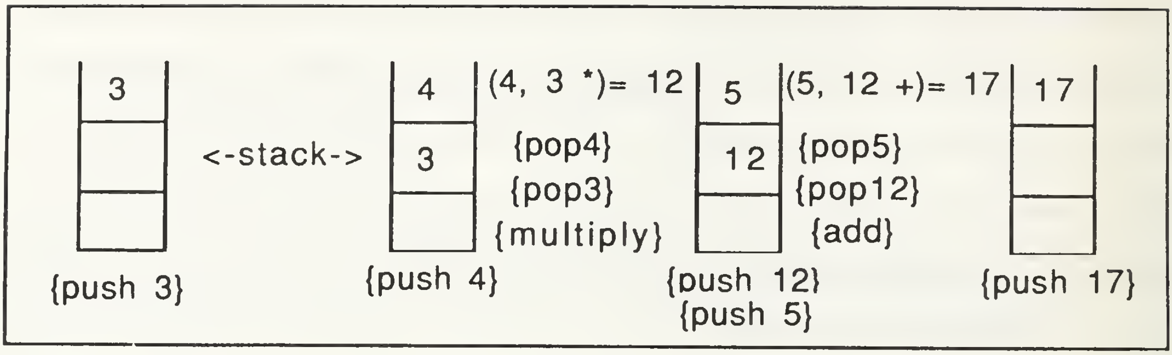

Figure 3.2 Example of the Stack Operation

(3) If the term is an operator then pop the first two items off the stack, apply the operator, and push the result back onto the top of the stack.

When the algorithm is completed, the answer to the expression will be on the top of the stack.

To get the infix equation into reverse Polish form is a bit more difficult than simply evaluating the Polish expression. Of special consideration when building the Polish form of the equation are the operator priorities and the use of parenthesis to change those priorities. A set of rules can be written which outline the conversion procedure [Ref. 2]. These rules are discussed below with an illustrative eaxmple. The infix expression

$$8 + (7 - 6 / 3) * 2$$

will be parsed using the following operator priority table.

Operator	Priority
^	4
*,/	3
+,−	2
operand	1
(,),space	0

Using a result string called RPN, and an operator stack the rules for infix to reverse Polish conversion follow:

(1) If an operand is encountered, move it to RPN.

(2) If an operator is encountered, move all higher priority operators on the stack to RPN and push the new operator onto the stack.

(3) If a left parenthesis is encountered push it onto the stack.

(4) If a right parenthesis is encountered, pop all operators off the stack and append them to RPN until a left parenthesis is encountered. Discard both parenthesis.

(5) When finished with the infix expression, pop all remaining operators from the stack and append them onto RPN.

Applying these rules to the example problem above is shown in Figure 3.3.

PRN	STACK	INFIX	RULE
		8 + (7 - 6 / 3) * 2	
8		+ (7 - 6 / 3) * 2	1
8	+	(7 - 6 / 3) * 2	2
8	+ (	7 - 6 / 3) * 2	3
8 7	+ (	- 6 / 3) * 2	1
8 7	+ (-	6 / 3) * 2	2
8 7 6	+ (-	/ 3) * 2	1
8 7 6	+ (- /	3) * 2	2
8 7 6 3	+ (- /	) * 2	1
8 7 6 3 / -	+	* 2	4
8 7 6 3 / -	+ *	2	2
8 7 6 3 / - 2	+ *		1
8 7 6 3 / - 2 * +			

Figure 3.3 Conversion from Infix to Reverse Polish

2. NextGain1 Function

This function makes two kind of intervals, linear interval and logarithmic interval. This function is called by the 'GetRoot1' procedure.

3. NextGain2 Function

This is like the 'NextGain1' function except it has two parameters. It supports two kind of intervals for each parameter. This function is called by the 'GetRoot2' procedure.

4. Results Procedure

This procedure outputs the calculated roots to the device 'Out File'. Then it makes it possible for the user to access the numerical data from the hard disk. Also, it supports the plot array named 'GraphArray' for the plotting data.

5. PlotRootLocus1 and PlotRootLocus2 Procedure

The structure of these procedures is exactly the same except for the auto–scale function of the coordinate to be supported by the 'PlotRootLocus1.' These procedures draw the roots and some information in the desired coordinate. First, the 'SelectWind' is called with the integer number and a boolean expression in order to select a window as visible. The 'Define Header' is called to draw the title.

Next the 'OpenPic' opens a picture for a specific window and only shows the drawing if the boolean is set True. In order to defind the coordinate it calls 'FindWorld' for the auto–scale or 'FindWorld1' for the manual–scale. Then 'DrawAxis' draws an axis with Footers and optional arrows on the axis. Finally, 'DrawPolygon' draws a polygon defined in the plot array with the predefined shape.

D. ROOTFINDER UNIT

The unit which solves for the roots of a polynomial is called 'RootFinder.'

This unit uses Laguerre's method with linear deflation. Since this method is a commercially available package of subroutines in the 'Turbo Pascal ToolBox (Numerical Methods)', a brief explanation is offered here.

1. Laguerre's Method

Laguerre's method attempts to approximate all the real and complex roots of a real or complex polynomial. Laguerre's method is very reliable and quick, even when converging to a multiple root.

To motivate (although not rigorously derive) the Laguerre formulas we can note the following relations between the polynomial and its roots and derivatives.

$$P_n(x) = (x - x_1)(x - x_2)\ldots(x - x_n) \tag{3.1}$$

$$\begin{aligned} \ln|P_n(x)| = \ln|x - x_1| + \ln|x - x_2| + \ldots \\ + \ln|x - x_n| \end{aligned} \tag{3.2}$$

$$\begin{aligned} \frac{d\ln|P_n(x)|}{dx} = \frac{1}{x - x_1} + \frac{2}{x - x_2} + \ldots \\ + \frac{1}{x - x_n} = \frac{\dot{P}_n}{P_n} \equiv G \end{aligned} \tag{3.3}$$

$$\begin{aligned} -\frac{d^2\ln|P_n(x)|}{dx^2} = \frac{1}{(x - x_1)^2} + \frac{1}{(x - x_2)^2} + \ldots \\ \frac{1}{(x - x_n)^2} = \left[\frac{\dot{P}_n}{P_n}\right] - \frac{\ddot{P}_n}{P_n} \equiv H \end{aligned} \tag{3.4}$$

Starting from these relations, the Laguerre formulas make what Acton nicely calls

"a rather drastic set of assumptions", the root x1 that we seek is assumed to be located some distance a from our current guess x, while all other roots are assumed to be located at a distance b.

$$a = x - x_1 \quad b = x - x_i \quad i = 2, 3, . . ., n \tag{3.5}$$

Then we can express Eqs (3.3), and (3.4) as

$$\frac{1}{a} + \frac{n - 1}{b} = G \tag{3.6}$$

$$\frac{1}{a^2} + \frac{n - 1}{b^2} = H \tag{3.7}$$

which yields as the solution for a

$$a = \frac{n}{G \pm \sqrt{(n - 1)(nH - G^2)}} \tag{3.8}$$

where the sign should be taken to yield the largest magnitude for the denominator. Since the factor inside the square root can be negative, a can be complex. (A more rigorous justification of Eq. (3.8) is in [Ref. 3].)

The method operates iteratively; for a trial value x, a is calculated by Eq. (3.8). Then x − a becomes the next trial value. This continues until a is sufficiently small. In the next section, a major procedure which handles this method will be explained.

2. InitAndTest Procedure

This procedure sets the initial value of the input and output variables. This procedure also tests the tolerance (Tol), maximum number of iterations (MaxIter), and code. Finally, it examines the coefficients of Poly. If the constant term is zero, then zero is one of the roots and the polynomial is deflated accordingly. Also if the leading coefficient is zero, the degree is reduced until the leading coefficient is non–zreo.

3. FindOneRoot Procedure

This procedure approximates a single root of the polynomial Poly. The root must be approximated within MaxIter iterations to a tolerance of Tol. The root, a value of the polynomial at the root, and the number of iterations(Iter) are returned. If no root is found, the appropriate error code (Error) is returned.

4. EvaluatePoly Procedure

This procedure applies the technique of synthetic division to determine the value (yValue), first derivative (yPrime) and second derivative (yDoublePrime) of the polynomial, Poly, at X. The 0th element of the first synthetic division is the value of Poly at X, the 1st element of the second synthetic division is the first derivative of Poly at X, and twice the 2nd element of the third synthetic division is the second derivative of Poly at X.

5. ConstructDifference Procedure

This procedure computes the difference between approximations; given information about the function and its first two derivatives.

6. TestForRoot Function

These are the stopping criteria. Four different ones are provided. If you wish to change the active criteria, simply comment off the current criteria

(including the appropriate OR) and remove the comment brackets from the criteria (including the appropriate OR) you wish to be active.

7. ReducePoly Procedure

This procedure deflates the polynomial Poly by factoring out the Root. Degree is reduced by one.

E. MESSAGE UNIT

This unit provides the several messages which inform the user with some warnings and help informations. There are 11 procedures,which have the same structure, to provide the content of a message and one procedure, 'SetupWindow,' to define the window. Here, the 'SetupWindow' procedure and one sample procedure to make message will be explained.

1. SetupWindow Procedure

This procedure defines all of the windows in the program. It calls the 'DefineWindow' procedure from 'TurboGraph' Unit. There are five other standard window types: documentPrc, DocProc, dBoxProc, PlainDBox, and noGrowDocProc [Ref. 4: P. 463]. We can insert window type, window ID, and size of window in this procedure.

2. MakeInfoScreen Procedure

This is one of nine procedures for messages. It is a sample to explain this kinds of procedure. 'MakeInfoScreen' procedure brings up a window with a description of what this program is. First, a 'GrafPort' is called. It is simply a Pascal record type with fields that control QuickDraw's behavior.[Ref. 4: p. 405 – 422] 'GrafPort' allows an application with windows to invoke drawing operations that are appropriate for each window. The 'SelectWind' chooses the window to be

defined in 'SetupWindow' using just window ID for this procedure. The selected window moves to the center of the screen using the 'Movewindow' and the 'SetVisibility' sets the visibility of a window. The 'TextFont,' 'TextSize,' and 'TextStyle' are used to define the letter to be drawn in the window. Finally the 'MoveTo' assigns the location to be drawn. 'DrawString' draws the messages in the window.

F. MYDIALOG UNIT

This unit supports 4 dialog boxes for input data. Each dialog procedure has the same skeleton to make a program; the only major procedure to construct the program will be explained in this section.

A dialog box communicates with the background text, controls, icons, and 'editText' items in which the user can enter and edit text; the user talks back with the mouse and keyboard.

The Dialog Manager [Ref. 5: p. 53] handles this communication. The Dialog manager leans heavily on the resource mechanism [Ref. 5: pp. 137 – 154] to provide it with data structures. Basically, each dialog is represented on disk as a resource of 'resType' DLOG–a template describing the dialog's size, window type, and title–and a resource of type DITL (Dialog Item List), which lists the content of the dialog (controls background text, and so on).

A program that intends to use dialogs sees to it that appropriate resources of type DLOG are available at runtime. Then, the program loads them into memory and draws them on the screen with 'GetNewDialog.'

After 'GetNewDialog' has read a dialog template and its item list into memory and drawn it on the screen, a program enters a loop in which it repeatedly

calls the lynchpin of dialog processing, 'ModalDialog,' and acts on the integer value it returns. 'ModalDialog' allows the user to customize its operation with a routine specified by the user. The user informs 'ModalDialog' of this by passing a specific parameter that points to the routine, which then gets control every time an event is generated when the dialog is on the screen. It is up to the filter routine to decide what to do about it.

To set the value of a control, you need to call 'SetCtlValue'. Since the 'SetCtlValue' expects a handle to the control record, you will have to get a handle to the control that needs setting. This is a task for the Dialog Manager's 'GetDItem' routine. 'GetDItem' takes in two bits of information and returns three. Given the indicated dialog and item number, it returns the information about that particular item: its type (such as 'radioButton' or 'staticText', encoded as an integer), a handle to its underlying data structure (for controls, this is a 'ControlHandle'), and finally, its bounding box. The key working with 'editText' items is to use 'GetIText' procedures. An 'editText' item begins life with the starting content assigned by its definition in an RMaker file [Ref. 5: p. 8]. After the user has played with it (as determined by a suitable 'itemHit' value returned by 'ModalDialog'), 'GetIText' is used to see what the value is now.

G. TURBOGRAPH UNIT

This unit supports many procedures for graphics under the Turbo Pascal environment. It has commercially available packages in the Turbo Pascal Toolbox (Numerical Method).

Many procedures and functions in this unit are called by several units. Since

these routines are already explained, they will not be explained again. The comments in source code of this unit give you enough information.

IV. EXAMPLE OF DESIGN

A. OVERVIEW

The root locus method is very valuable in the analysis of dynamic systems, and is also used for design. By inspection of the curve, we determine:

(1) Whether any loci cross the $j\omega$ axis into the right half s–plane (If such a crossing exists, it defines a stability limit for the system.),

(2) The frequency (value of ω) at such a stability limit,

(3) Whether the loci go through any area on the s–plane where we want dominant complex roots for our system (i.e., can we get what we want).

We can also determine the value of the parameter at the stability limit, the value of the roots for any specified value of the parameter, and the value of the parameter required to place roots at any selected point on the loci. These latter items, however, are not done by inspection of the curves, but by inspection of the computer printout or by separate calculations. On the MacRootLocus, both plot and calculation data are supported. Since two parameters of the system are adjustable, it is possible to calculate and plot a family of root loci for the pair of parameters The technique for synthesizing a system utilizing the rootlocus method is demonstrated in this chapter.

B. GRAPHICAL SOLUTION

In this section, four different kinds of control problems will be demonstrated. Such problems include simple cascade compensators for systems, subject only to

step inputs and/or load disturbances, and feedback compensators with no more than two adjustable variables.

1. Example 1 (cascade lead compensation)

The block diagram of Figure 4.1 shows the general case of cascade compensation.

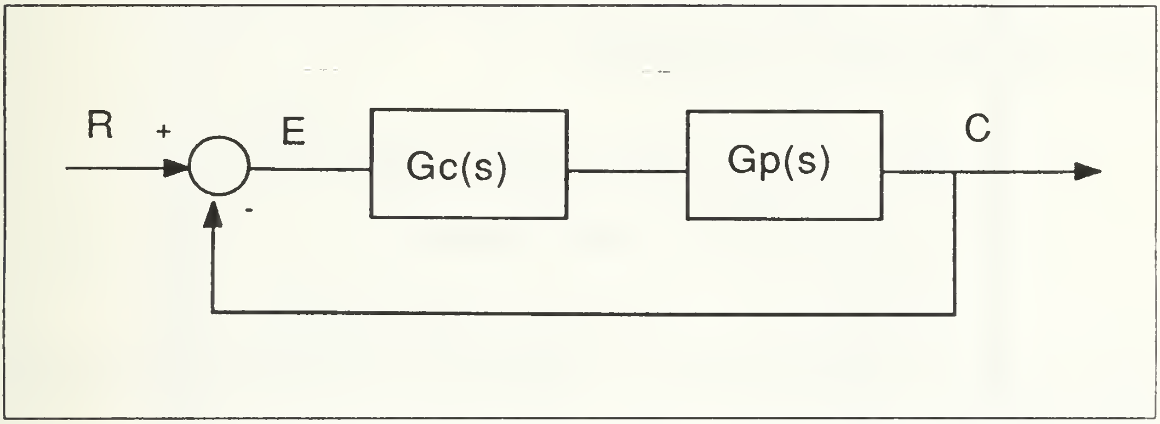

Figure 4.1 Simple Casade Compensation Block Diagram

Consider a plant with

$$G_P(S) = \frac{400(S + 1)}{S^2(S + 1)(S + 100)}. \tag{4.1}$$

The uncompensated system is unstable. It is desired to stabilize the system with a low–pass filter to reduce bandwidth. Specifications are now in the form of a desired location for the roots of the characteristic equation. A cascade lead compensator is to be used, and we choose the simplest possible i.e.,

$$G_c(S) = \frac{P\ (S\ +\ Z)}{Z\ (S\ +\ P)}. \tag{4.2}$$

Also we define A and B as

$$A = \frac{P}{Z}, \qquad B = P. \tag{4.3}$$

The characteristic equation for this system is

$$S^5 + (101 + B)S^4 + (100 + 101B)S^3 + (100B + 400A)S^2$$
$$+ 400A(1 + B/A)S + 400B = 0. \tag{4.4}$$

The two parameter root locus family for this system is given on Fig 4.2. Since the desired roots are

$$S = 1.684 \pm j\ 3.579,$$

an appropriate choice from this plot is A = 5.0, B = 4.8, from which Z = 0.96, and P = 5.0.

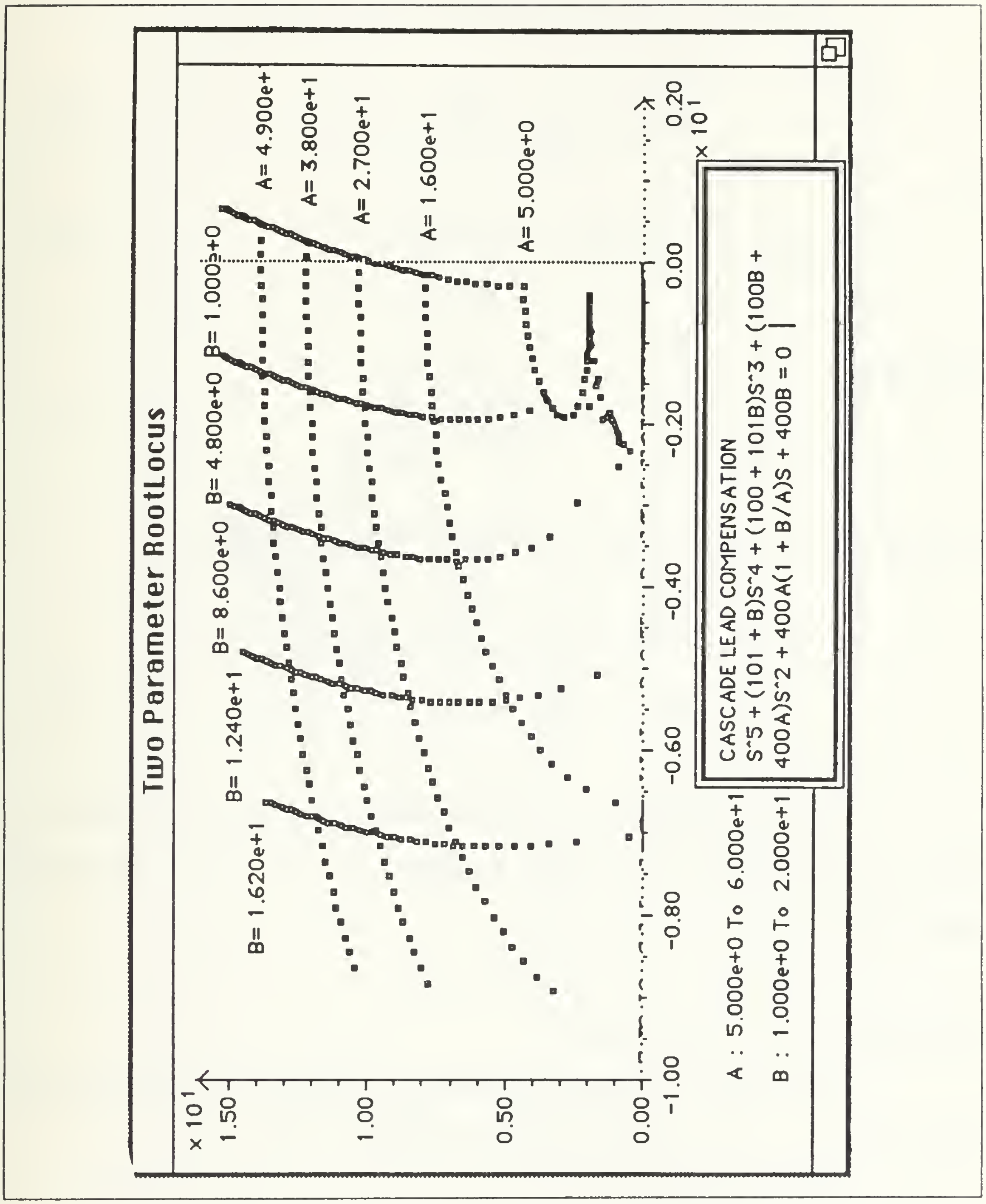

Figure 4.2 Plot of Example 1

2. Example 2 (cascade lag compensation)

If we consider a type–one third–order plant for which

$$G_p(S) = \frac{150}{S(S + 1)(S + 10)} \tag{4.5}$$

and

$$G_c(S) = \frac{P\ (S + Z)}{Z\ (S + P)}. \tag{4.6}$$

Then, letting

$$A = \frac{P}{Z}, \quad B = P, \tag{4.7}$$

the characteristic equation for the cascade lag compensated system is

$$S^4 + (11 + B)S^3 + (10 + 11B)S^2 + (10B + 150A)S + 150B = 0. \tag{4.8}$$

The two parameter root Locus for the lag relocation zone is shown on Figure 4.3. Choosing $A = 0.1$ and $B = 0.1$ gives $P = 0.01$, $Z = 0.1$ and provides complex roots at

$$S = 0.0372 \pm j\,1.12,$$

with real roots at $S = -1.014, -10.15$.

Stabilization can also be achieved with a lead filter, but the resulting system will have a wider bandwidth and faster response.

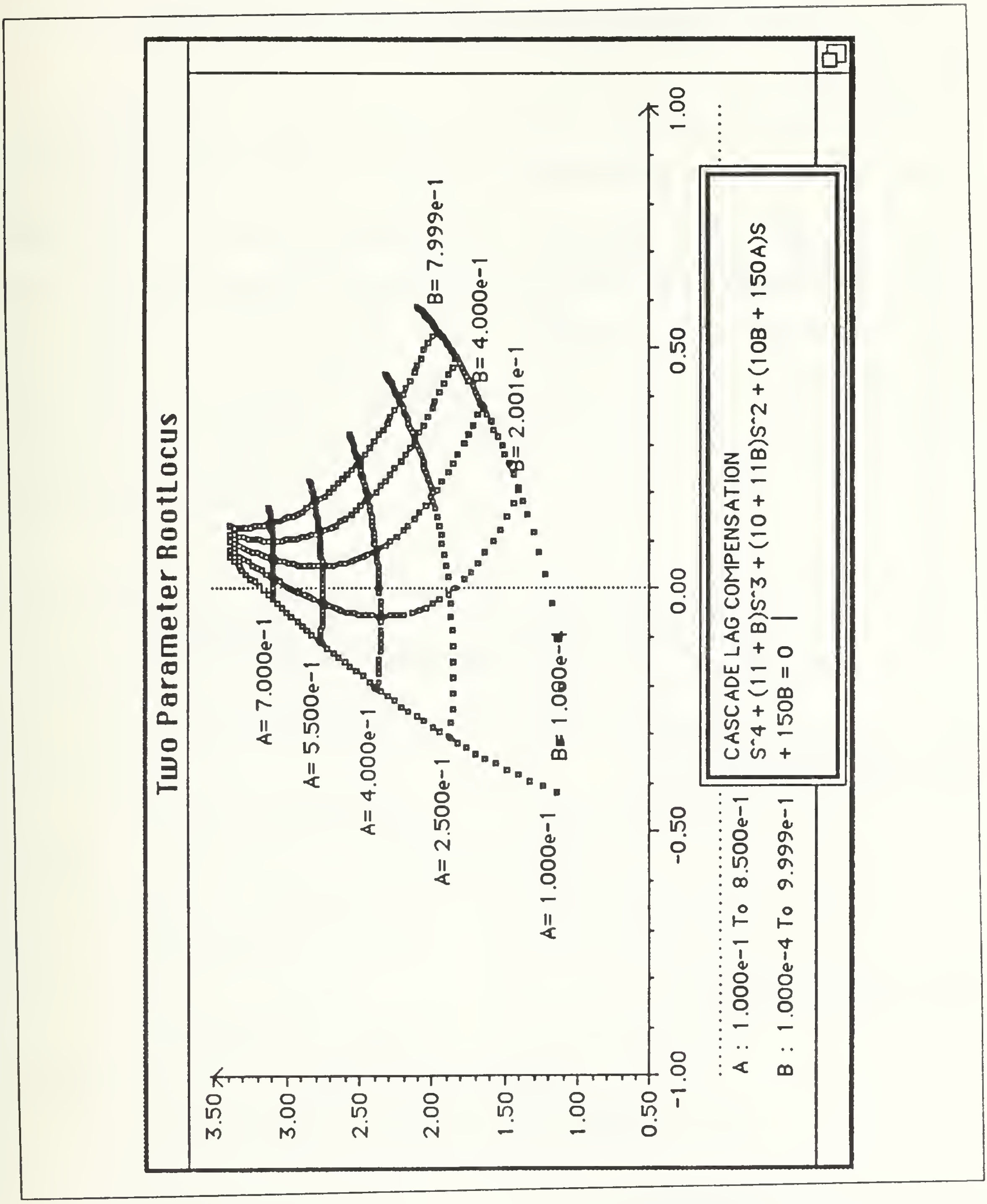

Figure 4.3 Plot of Example 2

3. Example 3 (velocity feedback)

The basic problem is given by the block diagram of Figure 4.4. G(s) may be any order.

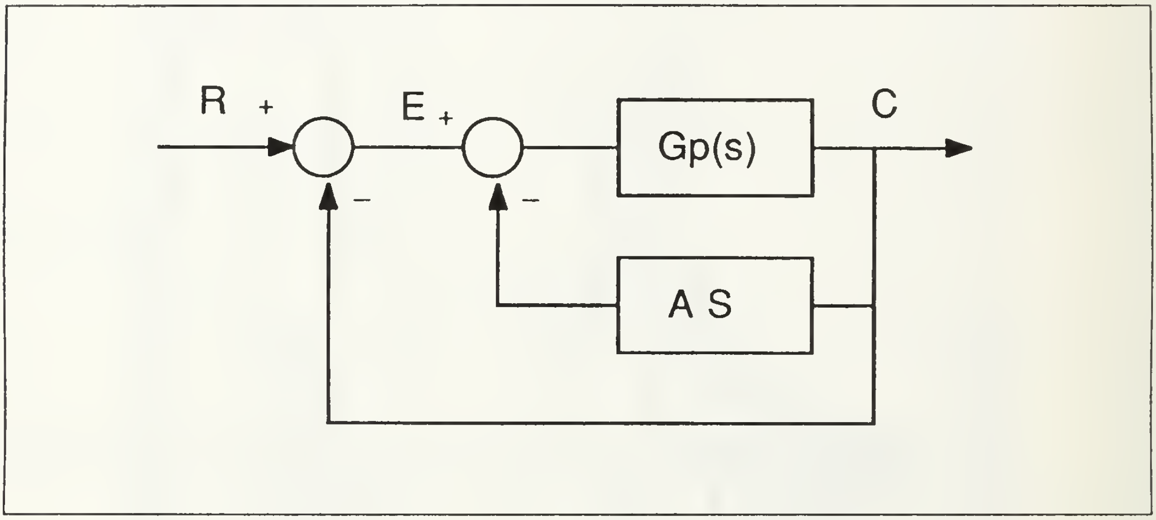

Figure 4.4 Velocity Feedback Compensation Block Diagram

For a second–order system, let

$$G(s) = 100 \;/\; S(S+2). \tag{4.9}$$

We want the root loci for

$$\frac{100\,AS}{S(S+2)+100} = -1, \tag{4.10}$$

so the characteristic equation is

$$S^2 + (2 + 100A)S + 100 = 0. \tag{4.11}$$

The resulting loci are shown on Fig 4.5. Inspection of Fig 4.5 shows that increasing the feedback gain increases damping. Any desired value of ζ is available for the complex roots. With high feedback gain, overdamping (all real roots) is available, and no positive value of gain can make the system unstable. To design the system, pick a root location on the locus, use the magnitude rule to find the gain as in Figure 4.5, or, since a computer program was used to generate the locus, the tabulated data are supplied to get the desired information. Part of the tabulated data are shown below.

```
 A =     0.07971
-4.98533003622524e+0     + 8.66870719484229e+0    j
-4.98533003622524e+0     +-8.66870719484229e+0    j

 A =     0.08633
-5.31655371460862e+0     + 8.46960781852863e+0    j
-5.31655371460862e+0     +-8.46960781852863e+0    j

 A =     0.09351
-5.67530563384649e+0     + 8.23352330186964e+0    j
-5.67530563384649e+0     +-8.23352330186964e+0    j

 A =     0.10128
-6.06387368606130e+0     + 7.95169390252752e+0    j
-6.06387368606130e+0     +-7.95169390252752e+0    j

 A =     0.10969
-6.48473591175408e+0     + 7.61237151975697e+0    j
-6.48473591175408e+0     +-7.61237151975697e+0    j

 A =     0.11881
-6.94057630317454e+0     + 7.19919443964476e+0    j
-6.94057630317454e+0     +-7.19919443964476e+0    j

 A =     0.12869
-7.43430192112068e+0     + 6.68813538631070e+0    j
-7.43430192112068e+0     +-6.68813538631070e+0    j
```

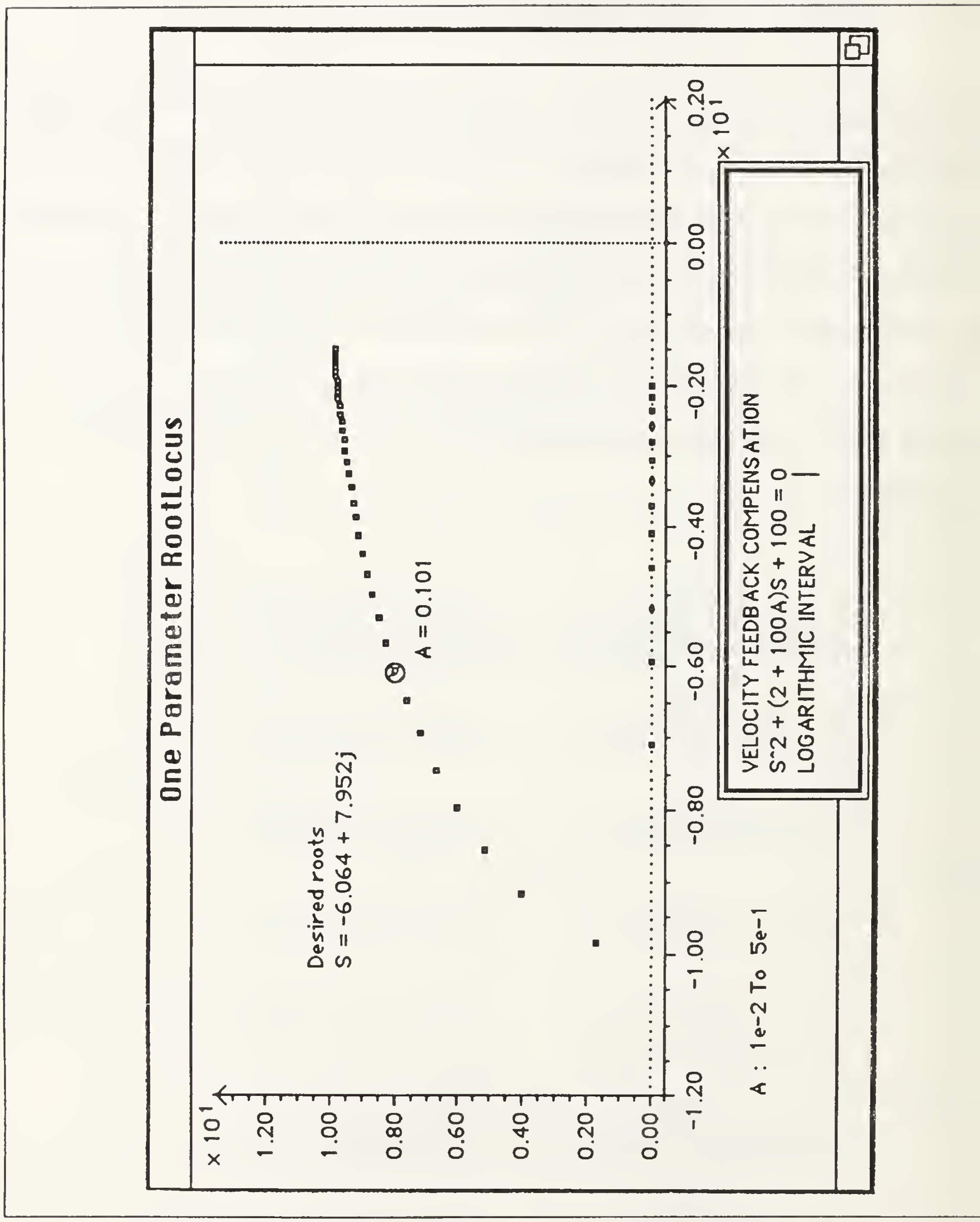

Figure 4.5 Plot of Example 3

4. **Example 4 (velocity and acceleration feedback)**

The block diagram of Figure 4.6 feedback shows the general case of velocity and acceleration feedback.

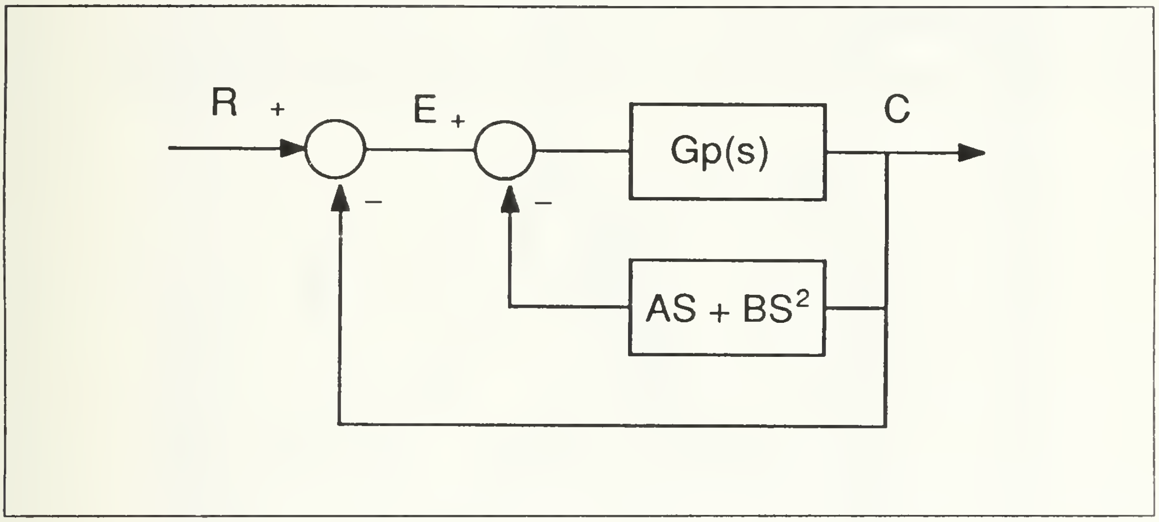

Figure 4.6 Velocity and Acceleration Feedback Compensation Block Diagram

If we consider a third order plant for which

$$G_P(s) = \frac{8}{(S + 1)^3}, \tag{4.12}$$

then the root locus form is

$$\frac{8(AS + BS^2)}{(S + 1)^3 + 8} = -1, \tag{4.13}$$

and the characteristic equation is

$$S^3 + (3 + 8A)S^2 + (3 + 8B)S + 9 = 0. \quad (4.14)$$

The two–parameter root locus family is given on Figure 4.7. We may select a desired location for the complex roots such as $s = -0.73 \pm j\,1.07$, for which $A_1 = 0.48$, $B_2 = 0.8$. The real root is at $s = -5.38$ so the complex roots are dominant.

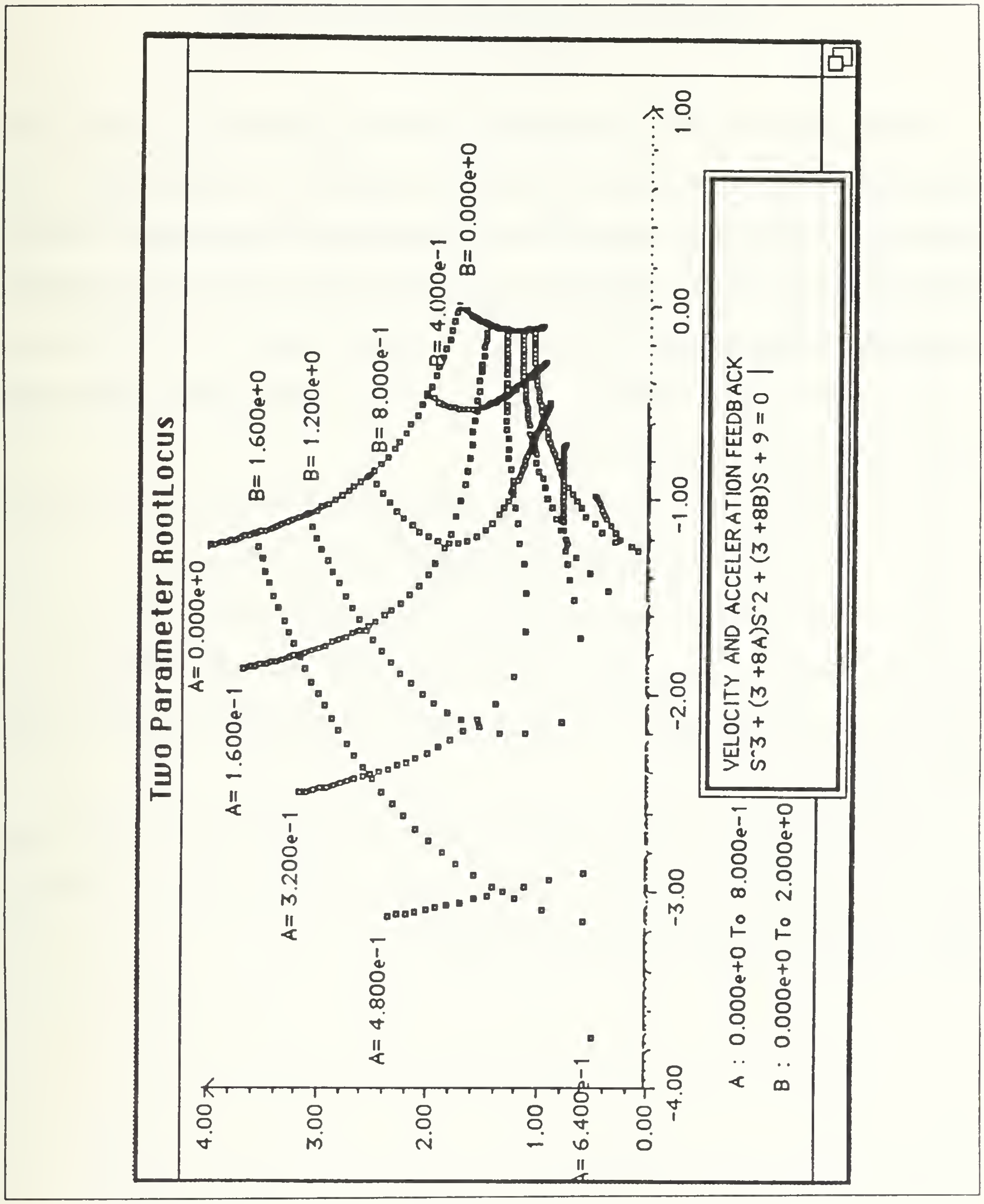

Figure 4.7 Plot of Example 4

V. CONCLUSION AND RECOMMENDATION

MacRootLocus is a useful and powerful tool for the designer and a simple and easy to use package for the user. The program supports the two parameter root locus method intensively as well as the one parameter root locus method. Also, it offers high resolution plots, the tabulated data and a small text editor for recording some information of the design.

The program does, however, have some areas where improvements are possible. These are discussed below.

(1) According to the engineer's needs, the root finder can be replaced easily with a more reliable and faster algorithm, and

(2) the capability to move the information window can be added.

Since this program was developed under the standard interface philosophy the Macintosh was designed for, it is a simple matter to change and to append some subroutines.

MacRootLocus is the user friendly CAD program available that is both simple enough for the beginning student to easily use as well as powerful and flexible enough to benefit the experienced system designer.

APPENDIX

SOURCE CODE

This appendix contains the source code of all modules in MacRootLocus except Standard Apple Macintosh libraries. Some subroutines in Numerical Method (Turbo Pascal Tool Box) are also included since they are slightly modified. A disk is available from Dr. Thaler that contains the MacRootLocus source code and the MacRootLocus resource file.

```
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{ =                                                                                      =}
{=                              MacRootLocus                                            =}
{=                               version1.0                                             =}
{ =                                                                                      =}
{=                              04  DEC  1989                                           =}
{ =                                                                                      =}
{=                                 author                                               =}
{=                             KO, SUNG HOON                                            =}
{=                                                                                      =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{ =                                                                                      =}
{=     SYSTEM      : Apple Macintosh SE                                                 =}
{=     LANGUAGE    : Turbo Pascal for Macintosh version1.0                              =}
{=     LIBRARIES   : Numerical Method (Turbo Pascal Tool Box)                           =}
{=     RESOURCE    : MacRootLocus.rsrc                                                  =}
{=                                                                                      =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

program MacRootLocus;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{= This main program handles the system and integrate theresource     =}
{=  and 6 units.                                                       =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

{$B+}                    { Set the bundle bit }
{$R MacRootLocus.Rsrc} { Identify resource file for menu and icon info }
{$T APPLFFTD}           { Set the application type and creator }
{$S+}                  { Generate segmented code}
{$I-}                    { Turn off I/O  error checking }
{$U SpecVar}
{$U  RootsFinder}
{$U  MakeRoot}
{$U  TurboGraph}
{$U Message}
{$U  MyDialog}

uses
     MemTypes, QuickDraw, OSIntf, ToolIntf,PackIntf,
     PasPrinter,SANE,MacPrint, RootsFinder, SpecVar,
   {$S Second Segment}
    TurboGraph,
   Message,
    {$S Third Segment}
   MakeRoot,
    MyDialog;
```

```
function UpCase(Ch : char) : char;
{=========================}
{=   Returns the upper case of Ch    =}
{=========================}
  inline
  $301F, { UpCase MOVE.W (SP)+,D0 ; GetCh }
   $0C40,
   $0061, { CMP.W #'a',D0 ; skip if not lower case }
  $6D0A, { BLT.S @1 }
   $0C40,
   $007A, { CMP.W #'z',D0 }
  $6E04, { BGT.S @1 }
   $0440,
   $0020, { SUB.W #$20,D0 }
  $3E80; { @1 MOVE.W D0,(SP) }

procedure HideAllWindows;
{= = = = = = = = = = = = = = = = = = =}
{= Hide the three main windows    =}
{= = = = = = = = = = = = = = = = = = =}
var
  WindNum : integer;
begin
   for WindNum :=  RegendBox downto InfoScrnWind do
    SetVisibility(WindNum, FALSE);
end; { HideAllWindows }

procedure ShowAllWindows;
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
{=   Make the three main windows visible      =}
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  WindNum : integer;
begin
  for WindNum := RootLocusWind to RegendBox do
  SelectWind(WindNum, TRUE);
end; { ShowAllWindows }

procedure MakeWatchCursor;
var
  CursorHandle : CursHandle;
begin
  CursorHandle := GetCursor(WatchCursor);
   SetCursor(CursorHandle^^);
end; { MakeWatchCursor }

procedure MakeArrowCursor;
```

```
begin
   InitCursor;
end; { MakeArrowCursor }

procedure  PrintScreen;
{= = = = = = = = = = = = = = = = = = = = = = = = =}
{=   Dump the entire screen to the printer.    =}
{= = = = = = = = = = = = = = = = = = = = = = = = }
begin
  HideCursor;
  HardCopy(FALSE);
  ShowCursor;
end; { PrintScreen }


procedure  DoDeskAcc(Item : Integer);
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
{=   start up desk accessory from Apple menu      =}
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  SavePort : GrafPtr;
  RefNum   : integer;
  DName    : string;
begin
  GetPort(SavePort);                    { save port before starting it  }
  GetItem(MenuList[AM], Item, DName);   { get name of desk accessory    }
  RefNum := OpenDeskAcc(DName);         { and start that sucker up!     }
  SetPort(SavePort);                    { restore grafport and continue }
end; { DoDeskAcc }

procedure SetItemState(Mndx,Indx : Integer; Flag : Boolean);
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=     if Flag is true, enables item Indx of menu Mndx; else disables      =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
begin
  if Flag then
     EnableItem  (MenuList[Mndx],Indx)
  else
     DisableItem(MenuList[Mndx],Indx);
end; { SetItemState }

procedure HandleMenu(MenuInfo  : LongInt);
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
{=   Decode MenuInfo and carry out command        =}
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  Menu : Integer;         { menu number that was selected     }
```

```
  Item : Integer;          { item in menu that was selected    }
  WindNum : integer;
begin
  if MenuInfo <> 0 then
  begin
    PenNormal;                    { set the pen back to normal       }
    Menu := HiWord(MenuInfo);        { find which menu the command is in }
    Item := LoWord(MenuInfo);       { get the command number            }
    case Menu of                  { and carry it out                 }
      ApplMenu  : if Item = 1 then
                    DoAbout           { bring up "About..." window}
                  else
                    DoDeskAcc(Item);  { start desk accessory      }

      FileMenu  : case Item of        { File menu }
                  1 : GetEQParameter;
                  2 : GetCoeff;
                    3 : PrintScreen;  {Print screen}
                  4 : begin
                        HideCursor;
                          HardCopy(TRUE); { Print top window }
                        ShowCursor;
                      end;
                  5 : Finished := TRUE; { Quit command }
                  end;

      EditMenu  : case Item of           { Edit Menu }
                    1..5 : if not SystemEdit(Item-1) then
                             { Do nothing }
                  end;

      PlotMenu  : case Item of             { bring up the dialog }
                    1 : PlotOneParameter;   {   for one parameter plot data}
                    2 : PlotTwoParameter;    {   for two parameter plot data}
                  end;

      HelpMenu  :
case Item of  { Help menu }
                  1 : InfoGetEQParameter;
                  2 : InfoGetCoeff;
                   3 : InfoPlotOneParameter;
                    4 : InfoPlotTwoParameter;
                    5 : InfoPrint
                  end

    end;{ case }
    HiliteMenu(0);                     { reset menu bar            }
```

```
  end;
end; { HandleMenu }

procedure SelectOurWindow(WPtr : WindowPtr);
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=    Select the window pointed to by WPtr if it's one of our's       =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
begin
  if (WPtr = Window[RootLocusWind].P) then
   SelectWind(RootLocusWind, TRUE);
end; { SelectOurWindow }

procedure HandleClick(WPtr : WindowPtr; MLoc : Point);
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
{=    Handle a mouse click within a window         =}
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
begin
  if WPtr <> FrontWindow then  { if it's not in front... }
    SelectOurWindow(WPtr);
end; { HandleClick }

procedure HandleGoAway(WPtr : WindowPtr; MLoc : Point);
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=    Handle a mouse click in the go-away box of a window     =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  WPeek : WindowPeek; { for looking at windows }
begin
  if WPtr = FrontWindow then      { if it's the active window }
   begin
     WPeek := WindowPeek(WPtr); { peek at the window }
     if TrackGoAway(WPtr, MLoc) then { and the box is clicked  }
    begin
       if WPeek^.WindowKind = userKind then { if it's our window }
        HideAllWindows                { time to stop  }
      else
        CloseDeskAcc(WPeek^.WindowKind) { close DeskAcc  }
    end
  end
 else
    SelectOurWindow(WPtr);
end; { HandleGoAway }

procedure HandleZoom(WPtr : WindowPtr; MLoc : Point; WLoc : integer);
{= = = = = = = = = =  = = = = = = = = = = = = = = = = =}
{=    Handle a mouse click in zoom box of a window     =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = =}
```

```
var
  WPeek : WindowPeek; { for looking at windows }
begin
  if WPtr = FrontWindow then { if it's the active window }
   begin
     WPeek := WindowPeek(WPtr); { peek at the window }
     if TrackBox(WPtr, MLoc, WLoc) then { and the box is clicked }
    begin
      if WPeek^.WindowKind = userKind then { if it's our window }
       ZoomWindow(WPtr, WLoc, FALSE);
    end
   end
  else
    SelectOurWindow(WPtr);
 end; { HandleZoom }

procedure HandleGrow(WPtr : WindowPtr; MLoc : Point);
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=      Handle mouse click in the grow box of a window          =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
type
  GrowRec = record
          case integer of
             0 : (Result : LongInt);
              1 : (Height, Width : integer);
        end;
var
  GrowInfo : GrowRec;
  WindNum  : integer;
begin
  if WPtr = FrontWindow then { if it's the active window }
   with GrowInfo do
   begin
     Result := GrowWindow(WPtr, MLoc, GrowArea); { get amount of growth }
     SizeWindow(WPtr, Width, Height, TRUE);     { resize window }
     InvalRect(WPtr^.portRect);                { set up for update }
     WindNum := 2 ;
     if Window[WindNum].P <> FrontWindow then
        DrawGrowIcon(Window[WindNum].P);        { draw grow icons }
   end
  else
    SelectOurWindow(WPtr);
end; { HandleGrow }

procedure HandleDrag(WPtr : WindowPtr; MLoc : Point);
{= = = = = = = = = = = = = = = = = = = = = = = =}
{=      Handle the dragging of a window      =}
```

```
{= = = = = = = = = = = = = = = = = = = = = = =}
var
  WindNum : integer;
begin
  if WPtr = FrontWindow then
   begin
      DragWindow(WPtr, MLoc, DragArea);  { in the drag bar }
     WindNum := 2 ;
      DrawGrowIcon(Window[WindNum].P); { draw grow icons }
   end
  else
     SelectOurWindow(WPtr);
end; { HandleDrag }

procedure DoMouseDown(TheEvent : EventRecord);
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=          identify where the mouse was clicked and handle it       =}
{= = = = = = = = = = = = = = = = = = = = = = =  = = = = = = = = =}
var
  theWindow :
WindowPtr;
  MLoc     :
Point;
  WLoc     :
integer;
begin
  MLoc  := TheEvent.Where;            { get mouse position }
  WLoc := FindWindow(MLoc, theWindow);{ get the window and the location  }

  case WLoc of
    InMenuBar   : HandleMenu(MenuSelect(MLoc));     { in the menu        }

    InContent   : HandleClick(theWindow, MLoc);     { inside the window  }

    InZoomIn    : HandleZoom(theWindow, MLoc, WLoc);{ in the zoom box    }

    InZoomOut   : HandleZoom(theWindow, MLoc, WLoc);{ in the zoom box    }

    InGoAway    : HandleGoAway(theWindow, MLoc);    { in the go away box }

    InGrow      : HandleGrow(theWindow, MLoc);      { in the grow box    }

    InDrag                                                                    :
HandleDrag(theWindow, MLoc);       { in the drag bar    }

    InSysWindow : SystemClick(TheEvent, theWindow); { in a DA window     }
  end;
```

```
end; { DoMouseDown }

procedure DoUpdate(TheEvent : EventRecord);
{= = = = = = = = = = = = = = = = = = = =}
{=    handles window update event    =}
{= = = = = = = = = = = = = = = = = = = =}
var
  SavePort,
  theWindow : WindowPtr;
begin
  theWindow := WindowPtr(TheEvent.Message);      { find which window }
  if (theWindow = Window[RootLocusWind].P) or
     (theWindow = Window[RegendBox].P)
     then             { only update our windows }
  begin
   MakeWatchCursor;
   GetPort(SavePort);                       { save current grafport  }
   SetPort(theWindow);                      { set as current port    }
   BeginUpdate(theWindow);                  { signal start of update }

   { and here's the update stuff! }

    if theWindow = Window[RootLocusWind].P then
     begin
       ClearWindow(RootLocusWind);
       DrawGrowIcon(theWindow);
       DrawPic(RootLocusWind);
     end;
    { now, back to our program...}

   EndUpdate(theWindow);                    { signal end of update  }
   SetPort(SavePort);                       { restore grafport      }
   MakeArrowCursor;                         { restore cursor        }
  end
end; { DoUpdate }

procedure DoActivate(TheEvent : EventRecord);
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
{=       Handles window activation event       =}
{= = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  AFlag     : boolean;
  theWindow : WindowPtr;
  WindNum   : integer;
begin
  with TheEvent do
  begin
```

```
    theWindow := WindowPtr(Message);            { get the window        }
    AFlag := Odd(Modifiers);                     { get activate/deactive }
    if AFlag then
     begin                              { if it's activated... }
       SetPort(theWindow);                       {   make it the port    }
      WindNum := RootLocusWind ;
       DrawGrowIcon(Window[WindNum].P);       {   draw grow icons       }
     end;
  end
end; { DoActivate }

procedure DoKeypress(theEvent : EventRecord);
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=        handles keypress (keyDown, autoKey) event            =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  KeyCh : char;
begin
  KeyCh := Chr(theEvent.Message and charCodeMask); { decode character }
  if (theEvent.modifiers and cmdKey) <> 0 then
    begin { menu key command }
      HandleMenu(MenuKey(KeyCh))                       { get menu and item}
    end
  else
   if TextInputEnabled then
     begin
       TEKey(KeyCh, textH);
        TEUpdate(thePort^.portRect, textH);
     end
    else
     SysBeep(1);                                { do *something*  }
end; { DoKeypress }

procedure  Initialize;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=         Initialize everything for the program           =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  Indx

integer;
  Result  : real;
   FileErr : byte;
begin                              { initialize all managers used      }
  InitGraf(@thePort);                  { create a grafport for the screen  }
  InitFonts;                       { start up the font manager          }
  InitWindows;                        { start up the window manager        }
```

```
InitMenus;                    { start up the menu manager        }
TEInit;
InitGraphic;
SetUpWindows;

TextFont(SystemFont);   { initialize the font   }

{ set up menus }
MenuList[AM] := GetMenu(ApplMenu);{ read menus in from resource fork  }
MenuList[FM] := GetMenu(FileMenu);
MenuList[EM] := GetMenu(EditMenu);
MenuList[PM] := GetMenu(PlotMenu);
MenuList[HM] := GetMenu(HelpMenu);

AddResMenu(MenuList[AM],'DRVR'); { pull in all desk accessories      }
for Indx := 1 to 5 do       { place menus in menu bar           }
   InsertMenu(MenuList[Indx],0);
for Indx := 1 to 6 do
  SetItemState(EM,Indx,FALSE);   { deactivate items in Edit menu     }
DrawMenuBar;                  { draw updated menu bar to screen  }
Finished := False;            { set program terminator to false   }

{ set drag region }
SetRect(DragArea, XMinGlb+1, YMinGlb+38, XMaxGlb-1, YMaxGlb-1);
{ set grow region }
SetRect(GrowArea, XMinGlb+1, YMinGlb+38, XMaxGlb-1, YMaxGlb-1);
InitGuess.Re := 1.0;
InitGuess.Im := 0.0;
Tolerance := 1e-6;
MaxIter := 100;
InitDegree := 0;
AMinGain := 0.1;
AMaxGain := 10000;
BMinGain := 0.1;
BMaxGain := 10000;
XMn     := -10 ;
XMx     := 5;
YMn     := -10;
YMx     := 10;
Step    := 4;
Points  := 50;
 FillChar(InitPoly, SizeOf(InitPoly), 0);
 FillChar(xAnswer, SizeOf(xAnswer), 0);
 FillChar(xInitPoly, SizeOf(xInitPoly), 0);
 FillChar(InfixArray, SizeOf(InfixArray), 0);
MakeWatchCursor;
MakeInfoScreen;
```

```
   MakeArrowCursor;
  TextInputEnabled := False;
  InitDegreeStatus := False;
 GetCoeffStatus   := False;

end; { Initialize }

procedure HandleEvent(TheEvent : EventRecord);
{= = = = = = = = = = = = = = = = = = = = = = = =}
{=          Decodes an event and handles it      =}
{= = = = = = = = = = = = = = = = = = = = = = = =}
begin
 case TheEvent.What of
   mouseDown
DoMouseDown(TheEvent);      { mouse button pushed    }
    keyDown      : DoKeyPress(TheEvent);       { key pressed down        }
    autoKey     : DoKeyPress(TheEvent);       { key held down           }
    updateEvt   : DoUpdate(TheEvent);        { window needs updating  }
    {activateEvt : DoActivate(TheEvent);}       { window made act/inact  }
 end
end; { HandleEvent }

procedure CleanUp;
{= = = = = = = = = = = = = = = = = = = = = = = = =}
{=          Do any last minute clean up work        =}
{= = = = = = = = = = = = = = = = = = = = = = = = =}
begin
end; { CleanUp }



begin { MacRootLocus }
 Initialize;                                   { set everything up          }
 repeat                                        { keep doing the following  }
   SystemTask;                                   { update desk accessories   }
    if GetNextEvent(everyEvent,theEvent) then  {    if there's an event...  }
     HandleEvent(theEvent);                      {      ...then handle it     }
 until Finished;                               { until user is done         }
 CleanUp;
end. { MacRootLocus }
```

```
unit GlobalVar (5000);
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=  This unit declares the whole variables tobe used      =}
{=  in MacRootLocus except a part of local variables.     =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

{$U RootsFinder}

interface

uses
 MemTypes,  QuickDraw, OSIntf, ToolIntf,PackIntf,PasPrinter,
   SANE,MacPrint, RootsFinder;

const
  InfoScrnWind  = 1;        { Window number of info window }
  RootLocusWind = 2;        { Window number of root locus window }
  RegendBox     = 3;        { Window number of informatiom box window }
  AboutBoxWind  = 4;        { Window number of about box window }
  AlertBox      = 5;        { Window number of alert box window }
  HelpWind      = 6;        { Window number of help window}

  MenuCnt       =    5;   { total # of menus                }
  ApplMenu      =  1000;  { resource ID of Apple Menu       }
  FileMenu      =  1001;  { resource ID of File Menu        }
  EditMenu      =  1002;  { resource ID of Edit Menu        }
  PlotMenu      =  1003;  { resource ID of Plot Menu        }
  HelpMenu      =  1004;  { resource ID of Help Menu        }

  AM          =    1;   { index into MenuList for Apple Menu }
  FM          =    2;   { ditto for File Menu              }
  EM          =    3;   { ditto for Edit Menu              }
  PM          =    4;   { ditto for Plot Menu              }
  HM          =    5;   { ditto for Help Menu              }

  MaxPlotGlb     = 1024;{ The maximum number of points in a plot array }

type
   str255 = string [255] ;
  StringArray = array[1..20] of str255;  { Storage for plot data }
  NodePtr = ^Node;
  Node =
      record
      Data : Extended;
       Next : NodePtr;
     end;
  PlotArray  = array[1..MaxPlotGlb, 1..2] of Extended;
```

```
var
  Answer, polish : str255;
   InfixArray  :  StringArray;
 A,B,ADeltaStep, AIncrem ,BDeltaStep, BIncrem: Extended;
 InitGuess : TNComplex;                  { Initial approximation }
 Tolerance : Extended;                 { Tolerance of approximation }
 AMaxGain, BMaxGain : Extended;         { Range of unknown parameter values }
 AMinGain, BMinGain : Extended;
 Root, Imag, Value, Deriv : TNvector;   { Resulting roots and other info }
 Iter : TNIntVector;                   { Iterations to find each root }
 MaxIter : integer;                     { Maximum number of iterations }
 InitDegree, Degree : integer;          { Initial and final degree }
 xInitPoly : TNvector;                               { of polynomial }
 InitPoly, Poly  : TNCompVector;              { Initial and final coefficients }
                                 { of the polynomial }
 xAnswer : TNCompVector;
 yAnswer : TNCompVector;
 NumRoots, ArrayIndex, pointno : integer;                  { Number of roots }
 Error : byte;                         { Error flag }
 StepA, StepB, linear : Boolean;
 Step,Points : Longint;

 OutFile     : text;                   { The output file used by an application}
 OutName     : string;                   { The out file name}
 IOerr       : boolean;                  { Flags I/O errors }

 Finished    : boolean;                 { used to terminate the program }
 theEvent    : EventRecord;               { event passed from operating system }
 MenuList    : array[1..MenuCnt] of MenuHandle; { holds menu info }
 DragArea  : Rect  ; { Area in which window can be dragged }
 GrowArea  : Rect;  { Area in which a window's size can change }

 DataPicture : PicHandle;               { Holds a picture of the plotted data }
 theDialog  : DialogPtr;
 itemHit    : Integer;
 theType    : Integer;
 r          : Rect;
 done, AutoScale      : Boolean;
 n           : Integer;
 h,h3,h4,h5,h6           : Handle;
 s           : Str255;
 XMn       : Extended;
 XMx       : Extended;
 YMn       : Extended;
 YMx       : Extended;

  GraphArray  :  ^PlotArray;
```

```
Ds  : DecStr ;
 ARJustification, AMarkStatus,BRJustification, BMarkStatus : Boolean;
Da, Db, Dan, Dax, Dbn, Dbx, Dss : DecStr;
txRect : Rect;
textH  : TEHandle;
TextInputEnabled : Boolean;
InitDegreeStatus : Boolean;
GetCoeffStatus   : Boolean;

implementation
begin
 end.{GlobalVar}
```

```
unit  MakeRoot(3000);

{= = = = = = = = = = = = = = = = = = = = = = = = == = = = = = = = = = = = =}
{=        This unit provides several procedures and functions to         =}
{=        make the roots and the array vectors for plot.                 =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = == = = = = = = = = = =}

{$I-}           { Disable I/O error trapping }
{$S+}          { Enable segmentation of code }
{$U SpecVar}
{$U Message}
{$U  RootsFinder}
{$U  TurboGraph}

interface

uses
  MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,PasPrinter,
  SANE, MacPrint,RootsFinder, SpecVar,

 {$S SecondSegment}
  TurboGraph,
 Message;

var
 Error  :  byte;

procedure GetRoot1;
procedure GetRoot2;

implementation

procedure  InfixToPolish(  Answer:str255;var  RPN:str255);
{= = = = = = = = = = = = = = = = = = = = = =  = = = = = = = = = = = = =  =}
{=        This procedure converts an algebraic expression(infix)        =}
{=      to reverse Polish notation.                                     =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = =  = = = = =  =}
 var
  Stack     : Array[1..50] of char;
  Top,I,N,p  : Integer;
  Ch       : char;
  Ch1       : string;
   Firstchar,PrevDigit   : Boolean;

  Function  Priority(Ch:char):  Integer;
  Begin
   Case Ch of
```

```
    '^'      : Priority:= 4;
    '*'      : Priority:= 3;
    '/'      : Priority:= 3;
    '-'      : Priority:= 2;
    '+'      : Priority:= 2;
    'a'..'b' : Priority:= 1;
    'A'..'B' : Priority:= 1;
    '0'..'9' : Priority:= 1;
    '.'      : Priority:= 1;
    '('      : Priority:= 0;
    ')'      : Priority:= 0;
    ' '      : Priority:= 0;
  end;
 end;

begin

 RPN := '';
 Top :=0;
 FirstChar :=True;
 PrevDigit := False;

 for i:= 1 to Length(Answer) do
   begin
     ch1 := copy(Answer,i,1);
     ch:= ch1[1];
      P:= Priority(ch);

     if Firstchar and (Ch = '-') then P:= 1;

    if P = 1 then
     begin
       if PrevDigit then
         RPN := concat(RPN,ch)
      else
         RPN := concat(RPN,' ',ch);
        firstchar := False; PrevDigit := True;
      end;
    if P>1 then
      begin

          while (Top>0) and (Priority(stack[top]) >= P) do
          begin
              RPN := concat(RPN,' ',Stack[Top]);
            Top := Top - 1;
          end;
       Top := Top + 1;
```

```
            Stack[Top] := ch;
              Firstchar := True; PrevDigit := False;
         end;
        if ch = '(' then
         begin
           Top := Top +1;
            Stack[Top] := '(';
             FirstChar := True; PrevDigit :=False;
         end;
        if ch = ')' then
         begin
           while Stack[Top]<> '(' do
               begin
                  RPN := concat(RPN,' ',Stack[Top]);
                 Top := Top -1;
               end;
           Top := Top -1;
            FirstChar := False; PrevDigit := False;
         end;
      end;
      while Top > 0 do
           begin
              RPN := concat(RPN,' ',Stack[Top]);
             Top := Top -1;
           end;
  end;

procedure ComputePolish(polish:str255;a,b:Extended; var EvalArray:Extended);
{= = = = = = = = = = = = = =  = = = =  = = = = = = = = = = = = = = = = = =}
{=       This procedure uses the string generated in Procedure           =}
{=        InfixToPolish to evaluate the numeric expression               =}
{= = = = = = = = = = = = = = = = = = = = = = = =  = = = = = = = = = = = = =}

 var
  i,N      : integer;
  ch,ch2   : char;
  ch1      : string;
  temp          : string[255];
  Value1,Value2,Value3 : Extended;
  StackPtr   : NodePtr;
  StackEmpty : Boolean;

    procedure CreateStack;
    {= = = = = = = = = = = = = =}
    {=       Initialize stack      =}
    {= = = = = = = = = = = = = =}
    begin
```

```
  New(StackPtr);
 with StackPtr^ do
 begin
  Next := nil;
  Data := 0.0;
 end;
 StackEmpty := True;
end;

procedure Pop(var Val : Extended);{Push a number onto numeric stack}
{= = = = = = = = = = = = = = = = = = = = = = =}
{=        pop a number off numeric stack      =}
{= = = = = = = = = = = = = = = = = = = = = = =}
var
 NPtr :NodePtr;
begin
 if not StackEmpty then
 begin
   NPtr := StackPtr^.Next;
   StackPtr^.Next := NPtr^.Next;
  Val := NPtr^.Data;
   Dispose(NPtr);
   StackEmpty := (StackPtr^.Next = nil);
 end;
end;

procedure Push(Val : Extended);
{= = = = = = = = = = = = = = = = = = = = = = = = =}
{=         pop a number off numeric stack         =}
{= = = = = = = = = = = = = = = = = = = = = = = = =}
var
 NPtr : NodePtr;
begin
 StackEmpty := False;
  New(NPtr);
 NPtr^.Data := Val;
 NPtr^.Next := StackPtr^.Next;
 StackPtr^.Next := NPtr;

end;

procedure DeleteStack;{Delete stack}
{= = = = = = = = = = =}
{=     Delete stack    =}
{= = = = = = = = = = =}
var
Temp : Extended;
```

```
begin
 while not StackEmpty do
  Pop(Temp);
  Dispose(StackPtr);
end;


 function Expon(y,x:Extended):Extended;
 {= = = = = = = = = = = = = = = = = = = = = = = =}
 {=        Computes Y raised to X power        =}
 {= = = = = = = = = = = = = = = = = = = = = = = =}
 begin
    Expon := exp( x * (ln(y)));
 end;


begin
  CreateStack;{Initialize}
 temp :=' ';

   for i:= 1 to Length(polish) do{do one char at a time}
   begin
     ch1 := copy(polish,i,1);
     ch:= ch1[1];{Get a char}

    case ch of                  {and evaluate it}
       '0'..'9' : temp :=concat(temp,ch);{Real constant}
      '.'      : temp := concat(temp,ch);
      '-'      : begin
                   ch1 := copy(polish,i+1,1);
                  ch2:= ch1[1];{Get a char}

                    if (ch2 <> ' ') and (i < length(polish))  then{Unary minus}
                    temp := concat(temp,ch)
                 else
                     begin        {minus operator}
                       POP(Value1);POP(Value2);
                     Value3 := Value2 - Value1;
                      PUSH(Value3);
                   end;
                end;

      'a'      : PUSH(a);
      'A'      : PUSH(A);
      'b'      : PUSH(b);
      'B'      : PUSH(B);
```

```
'+'     :begin
           POP(Value1); POP(Value2);
          Value3 := Value2 + Value1;

          PUSH(Value3);
        end;

'*'     :begin
           POP(Value1); POP(Value2);
          Value3 := Value2 * Value1;
          PUSH(Value3);

        end;


'/'     :begin
           POP(Value1); POP(Value2);
          Value3 := Value2 / Value1;
          PUSH(Value3);
        end;


'^'     :begin
           POP(Value1); POP(Value2);
           Value3 :=Expon( Value2 , Value1);
          PUSH(Value3);
        end;

' '     :begin
          if temp <> ' 'then
          begin
              Value1 := Str2Num(temp);
            PUSH(Value1);
            temp := ' ';
          end;
         end;

 end;
end;

if temp <> ' ' then
  begin
     Value3 := Str2Num(temp);
    PUSH(Value3);
  end;

 POP(Value1);
```

```
      EvalArray := Value1;
   DeleteStack;
   end;

function Ten2 (power: Extended) : Extended;
 begin
   Ten2 := Exp(power * Ln(10));
 end;

function NextGain1 : Extended;
{= = = = = = = = = = = = = = = = = =  = = = = = = = = = = = = = = = = = =}
{=     This function makes two kind of intervals for one                 =}
{=     parameter root locus method.                                      =}
{= = = = = = = = = = = = = = = = = = = = = =  = = = = = = = = = = = = = =}
 var
  GainOut, logmin, logmax : Extended;

 begin
  if linear then
  begin
     AIncrem := abs((AMaxGain - AMinGain)/(points - 1));
    GainOut := AMinGain + AIncrem * pointno;
  end
 else
  begin
    if (AMinGain  < 1E-5) and (AMaxGain > 1E-1) then
     logmin := -5
   else
      logmin := Ln(AMinGain)/Ln(10);
      logmax := ln(AMaxGain)/Ln(10);
      AIncrem := (logmax - logmin)/(Points -1);
     GainOut := Ten2(logmin + pointno * AIncrem);
  end;
  NextGain1 := GainOut;
 end;

function NextGain2 : Extended;
{= = = = = = = = =  = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=        This function makes two kind of intervals for each             =}
{=        parameter of two parameter root locus method.                  =}
{= = = = = = = = = = = = = = = = = = = = = = = = = =  = = = = = = = = = =}
 var
  GainOut, logmin, logmax : Extended;

 begin
  if StepA then
  begin
```

```
    if linea: then
    begin
       BIncrem := abs((BMaxGain - BMinGain)/(points - 1));
      GainOut := BMinGain + BIncrem * pointno;
    end
   else
    begin
      if (BMinGain  < 1E-5) and (BMaxGain > 1E-1) then
       logmin := -5
     else
        logmin := Ln(BMinGain)/Ln(10);
        logmax := ln(BMaxGain)/Ln(10);
         BIncrem := (logmax - logmin)/(Points -1);
       GainOut := Ten2(logmin + pointno * BIncrem);
    end;
   NextGain2 := GainOut;
  end
  else
  begin
   if linear then
   begin
      AIncrem := abs((AMaxGain - AMinGain)/(points - 1));
     GainOut := AMinGain + AIncrem * pointno;
   end
  else
   begin
     if (AMinGain  < 1E-5) and (AMaxGain > 1E-1) then
      logmin := -5
    else
       logmin := Ln(AMinGain)/Ln(10);
       logmax := ln(AMaxGain)/Ln(10);
        AIncrem := (logmax - logmin)/(Points -1);
       GainOut := Ten2(logmin + pointno * AIncrem);
   end;
  NextGain2 := GainOut;
  end;
 end;

procedure Results(A,B         : Extended;
                xAnswer    : TNCompVector;
                  Error      : byte);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=       This procedure outputs the results to the device OutFile        =}
{=      and make the array for plot data.                                =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
```

```
var
  Term : integer;

begin
  if Degree > 0 then
  begin
     Writeln(OutFile,'The deflated polynomial:');
   for Term := Degree downto 0 do
        Writeln(OutFile, 'Poly[',Term,']:',
                              Poly[Term].Re:23, ' +', Poly[Term].Im:23,'i');
  end;

   if Error <= 1 then
   begin
      Writeln(OutFile);
     Writeln(OutFile, ' A = ' , A :10:5,' B = ', B :10:5 );
    for Term := 1 to NumRoots do
    begin
         Writeln(OutFile, xAnswer[Term].Re:23, ' +', xAnswer[Term].Im:23, 'i');
       GraphArray^[ArrayIndex,1] := xAnswer[Term].Re;
        GraphArray^[ArrayIndex,2] := xAnswer[Term].Im;
      ArrayIndex := ArrayIndex + 1;
    end;
   end;
end; { procedure Results }

procedure PlotRootLocus1;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=          This procedure draw a plot of one parameter root locus method            =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
begin
   SelectWind(2, TRUE);
   DefineHeader(2, 'One Parameter RootLocus');
    DrawGrowIcon(Window[2].P);
   OpenPic(2, TRUE); { Open up a picture for window #1 }

    { Draw the axis and  data points }
    if AutoScale then
        FindWorld(2, GraphArray^, ArrayIndex)
    else
        FindWorld1(2,XMn,YMn,XMx,YMx);
     DrawAxis( Da , '', True);
      DrawPolygon(GraphArray^, 1, -ArrayIndex, -3, 1, 0, TRUE);
     ClosePicture; { Close the picture for window #2}
      ValidRect(Window[2].P^.portRect);
end; { PlotPowerExpLog }
```

```
procedure PlotRootLocus2;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=        This procedure draw a plot of two parameter root locus method        =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
begin
  SelectWind(2, TRUE);
  DefineHeader(2, 'Two Parameter RootLocus');
   DrawGrowIcon(Window[2].P);
  OpenPic(2, TRUE); { Open up a picture for window #2 }

   { Draw the axis and  data points }
      FindWorld1(2,XMn,YMn,XMx,YMx);
    DrawAxis(Da, Db, True);
     DrawPolygon(GraphArray^, 1, -ArrayIndex, -3, 1, 0, TRUE);
    ClosePicture; { Close the picture for window #2 }
     ValidRect(Window[2].P^.portRect);
end; { }

 procedure GetRoot1;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = }
{=        This procedure gets the root of the characteristic equation for       =}
{=       one parameter root locus method.                                       =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =  = = =}
  var
   term : integer;
  f : DecForm;
 begin
    Num2Str(f,AMinGain,Dan);
    Num2Str(f,AMaxGain,Dax);
   Da := 'A : '+ Dan + ' To '+Dax;
   Ds := ' ';
    ArrayIndex := 1;
    New (GraphArray);
   pointno := 0;
    while pointno <= ( Points - 1) do
    begin
     A := NextGain1;
     B := 0;
      for Term := InitDegree downto 0 do
       begin
             infixToPolish(InfixArray[Term],polish);
            ComputePolish(polish,a,b,xInitPoly[Term]);
           InitPoly[Term].Re := xInitPoly[Term];
          InitPoly[Term].Im := 0.0;
       end;
       Degree := InitDegree;
        Poly := InitPoly;
```

```
      Laguerre(Degree, Poly, InitGuess, Tolerance, MaxIter,
         NumRoots, xAnswer, yAnswer, Iter, Error);
      Results(A, B, xAnswer,  Error);
    pointno := pointno + 1;
   end;
    ArrayIndex := ArrayIndex - 1;
   PlotRootLocus1;
  end;

procedure GetRoot2{(var StepA : boolean; A,B : Extended)};
{= = = = = = = = = = = = = = = = = = =  = = = = = = =  = = = = = = = = = = = = = =}
{=          This procedure gets the root of the characteristic equation for =}
{=         two parameter root locus method.                                 =}
{= = = = = = = = = = = = = = = = = = = = = = = = = =  = = = =  = = = = = = = = =}
 var
  i, j, k, term : integer;
  f : DecForm;
 begin
   ADeltaStep := abs((AMaxGain - AMinGain)/(Step));
   BDeltaStep := abs((BMaxGain - BMinGain)/(Step));
   Num2Str(f,AMinGain,Dan);
   Num2Str(f,AMaxGain,Dax);
  Da := 'A : '+ Dan + ' To '+Dax;
   Num2Str(f,BMinGain,Dbn);
   Num2Str(f,BMaxGain,Dbx);
  Db := 'B : '+ Dbn + ' To '+Dbx;
   New (GraphArray);
  StepA := True;
  for i := 1 to 2 do
  begin
   for j := 1 to Step do
   begin
     ArrayIndex := 1;
     if StepA then A := AMinGain + ADeltaStep*(j-1)
            else B := BMinGain + BDeltaStep*(j-1);
     if StepA then
      begin
        Num2Str(f,A,Dss);
       Ds := 'A='+ Dss;
      end
     else
      begin
        Num2Str(f,B,Dss);
       Ds := 'B='+ Dss;
      end;
     pointno := 0;
     while pointno <= ( Points - 1) do
```

```
      begin
       if StepA then B := NextGain2
            else A := NextGain2;
        for Term := InitDegree downto 0 do
        begin
              infixToPolish(InfixArray[Term],polish);
             ComputePolish(polish,a,b,xInitPoly[Term]);
             InitPoly[Term].Re := xInitPoly[Term];
            InitPoly[Term].Im := 0.0;
        end;
        Degree := InitDegree;
         Poly := InitPoly;
          Laguerre(Degree, Poly, InitGuess, Tolerance, MaxIter,
                 NumRoots, xAnswer, yAnswer, Iter, Error);
          Results(A, B, xAnswer,  Error);
        pointno := pointno + 1;
       end;
         ArrayIndex := ArrayIndex - 1;
       PlotRootLocus2;
      end;
    StepA := False;
   end;
  end;
begin
end. { Unit MakeRoot }
```

```
unit  RootsFinder(2000);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=            Turbo Pascal Numerical Methods Toolbox                                =}
{=              Copyright (C) 1987 Borland International                            =}
{=                                                                                  =}
{=          This unit provides procedures for finding the roots                     =}
{=         of a single equation in one real variable.                               =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}


{$R+} { Enable range checking }

interface

uses
  MemTypes;

const
  TNNearlyZero = 1E-015;      { Close to zero }
  TNArraySize  = 10;          { Maximum size of vectors }

type
  TNvector      = array[0..TNArraySize] of Extended;

  TNIntVector   = array[0..TNArraySize] of integer;

  TNcomplex     = record
                Re, Im : Extended;
              end;

  TNCompVector = array[0..TNArraySize] of TNcomplex;


procedure Laguerre(var Degree     : integer;
                var Poly      : TNCompVector;
                    InitGuess : TNcomplex;
                    Tol       : Extended;
                     MaxIter   : integer;
                 var NumRoots  : integer;
                var Roots     : TNCompVector;
                var yRoots    : TNCompVector;
                 var Iter      : TNIntVector;
                 var Error     : byte);
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=                                                                                  =}
{=            Input: Degree, Poly, InitGuess, Tol, MaxIter                          =}
```

```
{=          Output: Degree, Poly, NumRoots, Roots, yRoots, Iter, Error          =}
{=                                                                               =}
{=          Purpose: This unit provides a procedure for finding all the          =}
{=                   roots (real and complex) to a polynomial.                   =}
{=                   Laguerre's method with deflation is used.                   =}
{=                   The user must input the coefficients of the quadratic      =}
{=                   and the tolerance in the answers generated.                 =}
{=                                                                               =}
{=          Pre-defined Types: TNcomplex    = record                             =}
{=                               Re, Im : Extended;                              =}
{=                             end;                                              =}
{=                                                                               =}
{=                 TNIntVector  = array[0..TNArraySize] of integer;              =}
{=                 TNCompVector = array[0..TNArraySize] of TNcomplex;            =}
{=                                                                               =}
{=          Variables: Degree    : integer;      degree of deflated             =}
{=                                         polynomial                            =}
{=                     Poly      : TNCompVector; coefficients of deflated       =}
{=                                         polynomial where Poly[n] is           =}
{=                                         the coefficient of X^n                =}
{=                     InitGuess : TNcomplex;    initial guess to a root         =}
{=                                         (may be very crude)                   =}
{=                     Tol       : Extended;     tolerance in the answer         =}
{=                     MaxIter   : integer;      number of iterations            =}
{=                     NumRoots  : integer;      number of roots calculated      =}
{=                     Roots     : TNCompVector; the roots calculated            =}
{=                     yRoots    : TNCompVector; the value of the function      =}
{=                                         at the calculated roots               =}
{=                     Iter      : TNIntVector;  number iteration it took to     =}
{=                                         find each root                        =}
{=                     Error     : byte;         flags an error                  =}
{=                                                                               =}
{=          Errors: 0: No error                                                  =}
{=                  2: Degree <= 0                                               =}
{=                  3: Tol <= 0                                                  =}
{=                  4: MaxIter < 0                                               =}
{=                                                                               =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

implementation

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=     The following inline procedure and function are used to call the user    =}
{=     defined procedures and functions pointed to by the ProcAddr parameter.   =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
```

```
function UserFunction(X : Extended; ProcAddr : ProcPtr) : Extended;
inline
  $205F,  { MOVE.L (A7)+, A0 }
  $4E90; { JSR    (A0)      }

procedure UserProcedure(X : TNcomplex; var Y : TNcomplex; ProcAddr : ProcPtr);
inline
  $205F,  { MOVE.L (A7)+, A0 }
  $4E90; { JSR    (A0)      }



procedure Laguerre{(var Degree    : integer;
                var Poly      : TNCompVector;
                   InitGuess : TNcomplex;
                  Tol       : Extended;
                    MaxIter   : integer;
                var NumRoots  : integer;
                var Roots     : TNCompVector;
                var yRoots    : TNCompVector;
                var Iter      : TNIntVector;
                  var Error     : byte)};

type
  TNquadratic = record
              A, B, C : Extended;
            end;

var
  AddIter     : integer;
  InitDegree  : integer;
  InitPoly    : TNCompVector;
  GuessRoot   : TNcomplex;

{-----------   Here  are  a  few  complex  operations  ------------}

procedure Conjugate(var C1, C2 : TNcomplex);
begin
  C2.Re := C1.Re;
   C2.Im := -1.0 * C1.Im;
end; { procedure Conjugate }

function Modulus(var C1 : TNcomplex) : Extended;
begin
   Modulus := Sqrt(Sqr(C1.Re) + Sqr(C1.Im));
end; { function Modulus }
```

```
procedure Add(var C1, C2, C3 : TNcomplex);
begin
 C3.Re := C1.Re + C2.Re;
  C3.Im := C1.Im + C2.Im;
end; { procedure Add }

procedure Sub(var C1, C2, C3 : TNcomplex);
begin
 C3.Re := C1.Re - C2.Re;
  C3.Im := C1.Im - C2.Im;
end; { procedure Sub }

procedure Mult(var C1, C2, C3 : TNcomplex);
begin
  C3.Re := C1.Re * C2.Re - C1.Im * C2.Im;
  C3.Im := C1.Im * C2.Re + C1.Re * C2.Im;
end; { procedure Mult }

procedure Divide(var C1, C2, C3 : TNcomplex);
var
  Dum1, Dum2 : TNcomplex;
 E : Extended;
begin
  Conjugate(C2, Dum1);
   Mult(C1, Dum1, Dum2);
   E := Sqr(Modulus(C2));
  C3.Re := Dum2.Re / E;
  C3.Im := Dum2.Im / E;
end;  { procedure Divide }

procedure SquareRoot(var C1, C2 : TNcomplex);
var
 R, Theta : Extended;
begin
   R := Sqrt(Sqr(C1.Re) + Sqr(C1.Im));
  if ABS(C1.Re) < TNNearlyZero then
    begin
      if C1.Im < 0 then
        Theta := Pi / 2
     else
          Theta := (-1.0 * Pi) / 2;
   end
  else
    if C1.Re < 0 then
       Theta := ArcTan(C1.Im / C1.Re) + Pi
    else
       Theta := ArcTan(C1.Im / C1.Re);
```

```
  C2.Re := Sqrt(R) * Cos(Theta / 2);
  C2.Im := Sqrt(R) * Sin(Theta / 2);
end; { procedure SquareRoot }

procedure InitAndTest(var Degree      : integer;
                      var Poly        : TNCompVector;
                          Tol         : Extended;
                          MaxIter     : integer;
                          InitGuess   : TNcomplex;
                      var NumRoots    : integer;
                      var Roots       : TNCompVector;
                      var yRoots      : TNCompVector;
                      var Iter        : TNIntVector;
                      var GuessRoot   : TNcomplex;
                      var InitDegree  : integer;
                      var InitPoly    : TNCompVector;
                      var Error       : byte);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=        Input:  Degree, Poly, Tol, MaxIter, InitGuess                          =}
{=        Output: InitDegree, InitPoly, Degree, Poly, NumRoots,                  =}
{=        Roots, yRoots, Iter, GuessRoot, Error                                  =}
{=                                                                               =}
{=       This procedure sets the initial value of the above                      =}
{=       variables.  This procedure also tests the tolerance                     =}
{=       (Tol), maximum number of iterations (MaxIter), and                      =}
{=       code.  Finally, it examines the coefficients of Poly.                   =}
{=       If the constant term is zero, then zero is one of the                   =}
{=       roots and the polynomial is deflated accordingly. Also                  =}
{=       if the leading coefficient is zero, then Degree is                      =}
{=       reduced until the leading coefficient is non-zero.                      =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =]

var
  Term : integer;

begin
  Error := 0;
  if Degree <= 0 then
    Error := 2;     { degree is less than 2 }
  if Tol <= 0 then
    Error := 3;
  if MaxIter < 0 then
    Error := 4;

  if Error = 0 then
  begin
```

```
    NumRoots := 0;
    GuessRoot := InitGuess;
    InitDegree := Degree;
    InitPoly := Poly;
    { Reduce degree until leading coefficient <> zero }
    while (Degree > 0) and (Modulus(Poly[Degree]) < TNNearlyZero) do
      Degree := Pred(Degree);
    { Deflate polynomial until the constant term <> zero }
    while (Modulus(Poly[0]) = 0) and (Degree > 0) do
    begin
      { Zero is a root }
      NumRoots := Succ(NumRoots);
      Roots[NumRoots].Re := 0;
      Roots[NumRoots].Im := 0;
      yRoots[NumRoots].Re := 0;
      yRoots[NumRoots].Im := 0;
      Iter[NumRoots] := 0;
      Degree := Pred(Degree);
      for Term := 0 to Degree do
        Poly[Term] := Poly[Term + 1];
    end;
  end;
end; { procedure InitAndTest }

procedure FindOneRoot(Degree     : integer;
                      Poly       : TNCompVector;
                      GuessRoot  : TNcomplex;
                      Tol        : Extended;
                      MaxIter    : integer;
                  var Root       : TNcomplex;
                  var yValue     : TNcomplex;
                  var Iter       : integer;
                  var Error      : byte);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=          Input:  Degree, Poly, GuessRoot, Tol, MaxIter                         =}
{=          Output: Root, yValue, Iter, Error                                     =}
{=                                                                                =}
{=   This procedure approximates a single root of the polynomial                  =}
{=   Poly.  The root must be approximated within MaxIter                          =}
{=   iterations to a tolerance of Tol.  The root, value of the                    =}
{=   polynomial at the root (yValue), and the number of iterations                =}
{=   (Iter) are returned. If no root is found, the appropriate error              =}
{=   code (Error) is returned.                                                    =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

var
```

```
  Found : boolean;
  Dif : TNcomplex;
  yPrime, yDoublePrime : TNcomplex;

procedure EvaluatePoly(Degree        : integer;
                       Poly          : TNCompVector;
                       X             : TNcomplex;
                   var yValue        : TNcomplex;
                   var yPrime        : TNcomplex;
                   var yDoublePrime : TNcomplex);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=  Input: Degree, Poly, X                                                          =}
{=  Output: yValue, yPrime, yDoublePrime                                            =}
{=  This procedure applies the technique of synthetic division to                   =}
{=  determine value (yValue), first derivative (yPrime) and second                  =}
{=  derivative (yDoublePrime) of the polynomial, Poly, at X.                        =}
{=  The 0th element of the first synthetic division is the                         =}
{=  value of Poly at X, the 1st element of the second synthetic                     =}
{=  division is the first derivative of Poly at X, and twice the                    =}
{=  2nd element of the third synthetic division is the second                       =}
{=  derivative of Poly at X.                                                        =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

var
  Loop : integer;
  Dummy, yDPdummy : TNcomplex;
  Deriv, Deriv2 : TNCompVector;

begin
  Deriv[Degree] := Poly[Degree];
  for Loop := Degree - 1 downto 0 do
  begin
    Mult(Deriv[Loop + 1], X, Dummy);
    Add(Dummy, Poly[Loop], Deriv[Loop]);
  end;
  yValue := Deriv[0];    { Value of Poly at X }

  Deriv2[Degree] := Deriv[Degree];
  for Loop := Degree - 1 downto 1 do
  begin
    Mult(Deriv2[Loop + 1], X, Dummy);
    Add(Dummy, Deriv[Loop], Deriv2[Loop]);
  end;
  yPrime := Deriv2[1];    { 1st deriv. of Poly at X }

  yDPdummy := Deriv2[Degree];
```

```
  for Loop := Degree - 1 downto 2 do
  begin
    Mult(yDPdummy, X, Dummy);
    Add(Dummy, Deriv2[Loop], yDPdummy);
  end;
  yDoublePrime.Re := 2 * yDPdummy.Re;    { 2nd derivative of Poly at X }
  yDoublePrime.Im := 2 * yDPdummy.Im;
end; { procedure EvaluatePoly }

procedure ConstructDifference(Degree       : integer;
                              yValue       : TNcomplex;
                              yPrime       : TNcomplex;
                              yDoublePrime : TNcomplex;
                          var Dif          : TNcomplex);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=  Input:  Degree, yValue, yPrime, yDoublePrime                               =}
{=  Output: Dif                                                                =}
{=                                                                             =}
{=   This procedure computes the difference between approximations;           =}
{=   given information about the function and its first two                   =}
{=   derivatives.                                                              =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

var
  yPrimeSQR, yTimesyDPrime, Sum, SRoot,
  Numer1, Numer2, Numer, Denom : TNcomplex;

begin
  Mult(yPrime, yPrime, yPrimeSQR);
  yPrimeSQR.Re := Sqr(Degree - 1) * yPrimeSQR.Re;
  yPrimeSQR.Im := Sqr(Degree - 1) * yPrimeSQR.Im;
  Mult(yValue, yDoublePrime, yTimesyDPrime);
  yTimesyDPrime.Re := (Degree - 1) * Degree * yTimesyDPrime.Re;
  yTimesyDPrime.Im := (Degree - 1) * Degree * yTimesyDPrime.Im;
  Sub(yPrimeSQR, yTimesyDPrime, Sum);
  SquareRoot(Sum, SRoot);
  Add(yPrime, SRoot, Numer1);
  Sub(yPrime, SRoot, Numer2);
  if Modulus(Numer1) > Modulus(Numer2) then
    Numer := Numer1
  else
    Numer := Numer2;
  Denom.Re := Degree * yValue.Re;
  Denom.Im := Degree * yValue.Im;
  if Modulus(Numer) < TNNearlyZero then
  begin
```

```
      Dif.Re := 0;
      Dif.Im := 0;
   end
  else
    Divide(Denom, Numer, Dif);  { The difference is the    }
                                 { inverse of the fraction }
end; { procedure ConstructDifference }

function TestForRoot(X, Dif, Y, Tol : Extended) : boolean;

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=   These are the stopping criteria.  Four different ones are                    =}
{=   provided.  If you wish to change the active criteria, simply                 =}
{=   comment off the current criteria (including the appropriate OR)              =}
{=   and remove the comment brackets from the criteria (including                 =}
{=  the appropriate OR) you wish to be active.                                     =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}


begin
  TestForRoot :=                                 {-------- ---------}
   (ABS(Y) <= TNNearlyZero)           {- Y=0                       -}
                                        {-                         -}
        or                             {-                          -}
                                        {-                         -}
   (ABS(Dif) < ABS(X * Tol))          {- Relative change in X     -}
                                        {-                         -}
                                        {-                         -}
 (*      or                  *)        {-                          -}
 (*                          *)        {-                          -}
 (* (ABS(Dif) < Tol)    *)             {- Absolute change in X     -}
 (*                          *)        {-                          -}
 (*      or                  *)        {-                          -}
 (*                          *)        {-                          -}
 (* (ABS(Y) <= Tol)    *)              {- Absolute change in Y     -}
                                                 {-----------------}

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=    The first criteria simply checks to see if the value of the                 =}
{=    function is zero.  You should probably always keep this criteria            =}
{=   active.                                                                       =}
{=                                                                                 =}
{=   The second criteria checks the relative error in X. This criteria            =}
{=  evaluates the fractional change in X between interations. Note                =}
{=   that X has been multiplied throught the inequality to avoid divide           =}
{=  by zero errors.                                                                =}
{=                                                                                 =}
```

```
{=  The third criteria checks the absolute difference in X between      =}
{=  iterations.                                                         =}
{=                                                                      =}
{=  The fourth criteria checks the absolute difference between          =}
{=  the value of the function and zero.                                 =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

end; { procedure TestForRoot }

begin { procedure FindOneRoot }
  Root := GuessRoot;
  Found := false;
  Iter := 0;
  EvaluatePoly(Degree, Poly, Root, yValue, yPrime, yDoublePrime);
  while (Iter < MaxIter) and not(Found) do
  begin
    Iter := Succ(Iter);
    ConstructDifference(Degree, yValue, yPrime, yDoublePrime, Dif);
    Sub(Root, Dif, Root);
    EvaluatePoly(Degree, Poly, Root, yValue, yPrime, yDoublePrime);
    Found := TestForRoot(Modulus(Root), Modulus(Dif), Modulus(yValue), Tol);
  end;
  if not(Found) then Error := 1;   { Iterations execeeded MaxIter }
end; { procedure FindOneRoot }

procedure ReducePoly(var Degree : integer;
                     var Poly   : TNCompVector;
                         Root   : TNcomplex);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=    Input: Degree, Poly, Root                                         =}
{=    Output: Degree, Poly                                              =}
{=                                                                      =}
{=    This procedure deflates the polynomial Poly by                    =}
{=    factoring out the Root.  Degree is reduced by one.                =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

var
  Term : integer;
  NewPoly : TNCompVector;
  Dummy : TNcomplex;

begin
  NewPoly[Degree - 1] := Poly[Degree];
  for Term := Degree - 1 downto 1 do
  begin
    Mult(NewPoly[Term], Root, Dummy);
```

```
    Add(Dummy, Poly[Term], NewPoly[Term - 1]);
  end;
  Degree := Pred(Degree);
  Poly := NewPoly;
end; { procedure ReducePoly }

begin  { procedure Laguerre }
  InitAndTest(Degree, Poly, Tol, MaxIter, InitGuess, NumRoots, Roots,
              yRoots, Iter, GuessRoot, InitDegree, InitPoly, Error);
  while (Degree > 0) and (Error = 0) do
  begin
    FindOneRoot(Degree, Poly, GuessRoot, Tol, MaxIter,
                Roots[NumRoots + 1], yRoots[NumRoots + 1],
                Iter[NumRoots + 1], Error);
    if Error = 0 then
    begin
      {= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
      {=  The next statement refines the approximate root by            =}
      {=  plugging it into the original polynomial.  This               =}
      {=  eliminates a lot of the round-off error                       =}
      {=  accumulated through many iterations                           =}
      {= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
      FindOneRoot(InitDegree, InitPoly, Roots[NumRoots + 1],
                  Tol, MaxIter, Roots[NumRoots + 1],
                  yRoots[NumRoots + 1], AddIter, Error);
      Iter[NumRoots + 1] := Iter[NumRoots + 1] + AddIter;
      NumRoots := Succ(NumRoots);
      ReducePoly(Degree, Poly, Roots[NumRoots]); { Reduce polynomial }
    end;
    GuessRoot := Roots[NumRoots];
  end;
end; { procedure Laguerre }

begin
end. { RootsOfEquat }
```

```
unit Message (6000);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = == = = = =}
{=        This unit provides the several messages which inform the        = }
{=        user with some warnings and the help informations.              =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =  =  = = =}

{$T APPLFFTD}     { Set the application type and creator }
{$S+}
{$I-}            { Turn off I/O  error checking }
{$U RootsFinder}
{$U SpecVar}
{$U TurboGraph}

interface
uses
    MemTypes, QuickDraw, OSIntf, ToolIntf,PackIntf,
    PasPrinter,SANE,MacPrint, RootsFinder, SpecVar,
   {$S Second Segment}
   TurboGraph;

procedure SetUpWindows;
procedure MakeInfoScreen;
procedure DoAbout;
procedure  AlertBox1;
procedure  AlertBox2;
procedure  AlertBox3;
procedure InfoGetEQParameter;
procedure InfoGetCoeff;
procedure InfoPlotOneParameter;
procedure InfoPlotTwoParameter;
procedure  InfoPrint;

implementation

procedure SetUpWindows;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=         Define all of the windows used in the program       =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
begin
  { The initial information screen }
   DefineWindow(InfoScrnWind, 100, 100, 362, 200, dBoxProc);

  { The real transform window }
     DefineWindow(RootLocusWind,  XMinGlb+5,  YMinGlb+43,  XMaxGlb-5,  YMaxGlb-5,
documentProc);
  DefineHeader(RootLocusWind, 'Root Locus');
```

```
  DefineWindow(RegendBox,100, 100,362,150,  dBoxProc);

 { The about box window }
  DefineWindow(AboutBoxWind, 125, 60, 387, 272, dBoxProc);
  DefineWindow(AlertBox,50, 100,360,170,  dBoxProc);
  DefineWindow(HelpWind,20,60,480,330,dBoxProc);

end; { SetUpWindows }

procedure MakeInfoScreen;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=      Bring up a window with a description of what this program does      =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }

begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(InfoScrnWind, FALSE);
  with Window[InfoScrnWind].P^.portRect do
 begin
    InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
    InTop := (YMaxGlb - (Bottom-Top)) div 2;
 end;
  MoveWindow(Window[InfoScrnWind].P, InLeft, InTop, TRUE); { Center the window }
  SetVisibility(InfoScrnWind, TRUE);
  TextFont(SystemFont);
  TextSize(14);
  TextStyle([]);
  MoveTo(20, 20);
  DrawString('Welcome To');
  TextSize(16);
  MoveTO(70, 50);
  DrawString('MAC RootLocus');
  TextSize(12);
  TextStyle([]);
  MoveTo(90, 80);
  DrawString('Have a fun !!');
  repeat until Button;
  HideWindow(Window[InfoScrnWind].P);
  GetPort(SavePort); { save the current grafport }
end; { MakeInfoScreen }
```

```
procedure DoAbout;
{= = = = = = = = = = = = = = = = = = =}
{=       Bring up the about box       =}
{= = = = = = = = = = = = = = = = = = =}
var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }

begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(AboutBoxWind, FALSE);
  with Window[AboutBoxWind].P^.portRect do
  begin
    InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
    InTop := (YMaxGlb - (Bottom-Top)) div 2;
  end;
  MoveWindow(Window[AboutBoxWind].P, InLeft, InTop, TRUE); { Center the window }
  SetVisibility(AboutBoxWind, TRUE);
  TextFont(SystemFont);
  TextSize(18);
  TextStyle([]);
  MoveTo(60, 40);
  DrawString('MAC RootLocus');
  TextStyle([]);
  TextSize(12);
  MoveTo(85, 60);
  DrawString('version 1.00');
  MoveTo(35, 90);
  DrawString('Computer Aided Desgin Tool');
  MoveTo(35, 110);
  DrawString('For RootLocus Analysis');
  MoveTo(97, 160);
  DrawString('written by');
  TextSize(13);
  MoveTo(85, 185);
  DrawString('KO SUNG HOON');
  TextFont(SystemFont);
  repeat until Button;
  HideWindow(Window[AboutBoxWind].P);
  SetPort(SavePort);
end; { DoAbout }

procedure AlertBox1;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=     Bring up a window with a description of warning message.           =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
```

```
var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }
 begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(3, FALSE);
  with Window[3].P^.portRect do
  begin
    InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
    InTop := (YMaxGlb - (Bottom-Top)) div 2;
  end;
  MoveWindow(Window[3].P, InLeft, InTop, TRUE); { Center the window }
  SetVisibility(3, TRUE);
  TextFont(SystemFont);
  TextSize(12);
  TextStyle([]);
  MoveTo(30, 30);
  DrawString('There is no Initial Degree.');
  repeat until Button;
  HideWindow(Window[3].P);
 end;


procedure AlertBox2;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = }
{=      Bring up a window with a description of warning message.      =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = }
var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }
 begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(5, FALSE);
  with Window[5].P^.portRect do
  begin
    InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
    InTop := (YMaxGlb - (Bottom-Top)) div 2;
  end;
  MoveWindow(Window[5].P, InLeft, InTop, TRUE); { Center the window }
  SetVisibility(5, TRUE);
  TextFont(SystemFont);
  TextSize(12);
  TextStyle([]);
  MoveTo(30, 30);
```

```
   DrawString('There is no Initial Degree or');
   MoveTo(50,50);
   DrawString('Characteristic Equation Coefficients.');
  repeat until Button;
   HideWindow(Window[5].P);
 end;

procedure AlertBox3;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=       Bring up a window with a description of warning message.        =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }
 begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(5, FALSE);
  with Window[5].P^.portRect do
  begin
    InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
    InTop := (YMaxGlb - (Bottom-Top)) div 2;
  end;
  MoveWindow(Window[5].P, InLeft, InTop, TRUE); { Center the window }
  SetVisibility(5, TRUE);
  TextFont(SystemFont);
   TextSize(12);
   TextStyle([]);
  MoveTo(30, 30);
  DrawString('You have a wrong Input.');
   MoveTo(50,50);
  DrawString('Check the Help Menu and try again.');
  repeat until Button;
   HideWindow(Window[5].P);
 end;

procedure InfoGetEQParameter;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=          Bring up the Help information of EQ parameter dialog.        =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }

begin
  GetPort(SavePort); { save the current grafport }
```

```
 SelectWind(HelpWind, FALSE);
 with Window[HelpWind].P^.portRect do
begin
   InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
   InTop := (YMaxGlb - (Bottom-Top)) div 2;
end;
 MoveWindow(Window[HelpWind].P, InLeft, InTop, TRUE); { Center the window }
 SetVisibility(HelpWind, TRUE);
 TextFont(SystemFont);
 TextSize(14);
 TextStyle([]);
 MoveTo(150, 15);
 DrawString('EQ PARAMETER');
 TextStyle([]);
 TextSize(12);
 MoveTo(20, 40);
 DrawString('The EQ Parameter stands for equation parameter. It will allow you');
 MoveTo(10, 55);
 DrawString('to input the degree of polynomial and some equation parameters. ');
 MoveTo(10,70);
 DrawString('The degree of the polynomial should be 1 up to 10. Then there are  ');
 MoveTo(10, 85);
 DrawString('the default values for the other parameters. These values avoid ');
 MoveTo(10, 100);
 DrawString('the convergence error for almost all polynomials. But if ');
 MoveTo(10, 115);
 DrawString('convergence error messages appears on the screen, you can change ');
 MoveTo(10, 130);
 DrawString('these parameter values. These parameters must satisfy the ');
 MoveTo(10, 145);
 DrawString('following conditions:');
 MoveTo(30, 160);
 DrawString('(1) Initial guess >= 0 ');
 MoveTo(30, 175);
 DrawString('(2) Maximum Iteration >= 0 ');
 MoveTo(30, 190);
 DrawString('(3) Tolerance > 0 ');
 MoveTo(40, 260);
 DrawString('" CLICK THE MOUSE ONCE TO RETURN THE MAIN MENU "');
 repeat until Button;
 HideWindow(Window[HelpWind].P);
 SetPort(SavePort);
end;

procedure InfoGetCoeff;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=           Bring up the Help information of Get Coeff dialog.          =}
```

```
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }

begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(HelpWind, FALSE);
  with Window[HelpWind].P^.portRect do
  begin
    InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
    InTop := (YMaxGlb - (Bottom-Top)) div 2;
  end;
  MoveWindow(Window[HelpWind].P, InLeft, InTop, TRUE); { Center the window }
  SetVisibility(HelpWind, TRUE);
  TextFont(SystemFont);
  TextSize(14);
  TextStyle([]);
  MoveTo(170, 15);
  DrawString('GET COEFF');
  TextStyle([]);
  TextSize(12);
  MoveTo(20, 40);
  DrawString('The Get Coeff stands for Get Coefficients. It will allow you to');
  MoveTo(10, 55);
  DrawString('input the algebraic expression for the coefficients of the    ');
  MoveTo(10,70);
  DrawString('characteristic equation. It may have up to two undetermined  ');
  MoveTo(10, 85);
  DrawString('parameters(A and B). In case of the one parameter root locus ');
  MoveTo(10, 100);
  DrawString('method, you use only one undetermined paramerter(A). The routine');
  MoveTo(10, 115);
  DrawString('uses standard algebraic, or infix, notation with parenthesis');
  MoveTo(10, 130);
  DrawString('allowed. Operater can include +, -, *, /, and ^(expomentiation).');
  MoveTo(10, 145);
  DrawString('The unary minus sign is allowed.');
  MoveTo(10, 160);
  DrawString('If you choose the Get Coeff command without the degree of the');
  MoveTo(10, 175);
  DrawString('polynomial,the message appears on the screen. It tells you that');
  MoveTo(10, 190);
  DrawString('the degree of the polynomial has not been entered yet.');
  MoveTo(40, 260);
  DrawString('" CLICK THE MOUSE ONCE TO RETURN THE MAIN MENU "');
```

```
  repeat until Button;
   HideWindow(Window[HelpWind].P);
   SetPort(SavePort);
end;

procedure InfoPlotOneParameter;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=        Bring up the Help information of One parameter dialog.        =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }

begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(HelpWind, FALSE);
  with Window[HelpWind].P^.portRect do
 begin
    InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
    InTop := (YMaxGlb - (Bottom-Top)) div 2;
 end;
  MoveWindow(Window[HelpWind].P, InLeft, InTop, TRUE); { Center the window }
   SetVisibility(HelpWind, TRUE);
  TextFont(SystemFont);
   TextSize(14);
    TextStyle([]);
   MoveTo(145, 15);
  DrawString('ONE PARAMETER');
    TextStyle([]);
   TextSize(12);
   MoveTo(20, 35);
  DrawString('One parameter command will allow you to input the plot data of');
   MoveTo(10, 50);
  DrawString('the one parameter root locus method. The default values are shown');
   MoveTo(10,65);
  DrawString('in the dialog box. They can be changed as desired.');
   MoveTo(20, 80);
  DrawString('-. Enter the minimum and maximum gain values into the Min Gain');
   MoveTo(10, 95);
  DrawString('and the Max Gain insertion box.');
   MoveTo(20, 110);
  DrawString('-. Choose the type of interval. There are two types of interval:');
   MoveTo(10, 125);
  DrawString('Linear and Logarithmic interval. When you click the radio button,');
   MoveTo(10, 140);
  DrawString('the desired type of interval is choosen.');
```

```
 MoveTo(20, 155);
 DrawString('-.Choose the type of scale for the axis. There are two types of  ');
 MoveTo(10, 170);
 DrawString('scale: Auto and Manual scale. When you click the radio button,');
 MoveTo(10, 185);
 DrawString('the desired type of scale is choosen.');
 MoveTo(20, 200);
 DrawString('-.Enter some number into Points to Plot inserton box in order to ');
 MoveTo(10, 215);
 DrawString('decide the plot resolution.');
 MoveTo(40, 260);
 DrawString('" CLICK THE MOUSE ONCE TO RETURN THE MAIN MENU "');
 repeat until Button;
 HideWindow(Window[HelpWind].P);
 SetPort(SavePort);
end;

procedure InfoPlotTwoParameter;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = }
{=        Bring up the Help information of the two parameter dialog.          =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = }
var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }

begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(HelpWind, FALSE);
  with Window[HelpWind].P^.portRect do
 begin
    InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
    InTop := (YMaxGlb - (Bottom-Top)) div 2;
 end;
  MoveWindow(Window[HelpWind].P, InLeft, InTop, TRUE); { Center the window }
  SetVisibility(HelpWind, TRUE);
  TextFont(SystemFont);
  TextSize(14);
  TextStyle([]);
  MoveTo(145, 15);
  DrawString('TWO PARAMETER');
  TextStyle([]);
  TextSize(12);
  MoveTo(20, 35);
  DrawString('Two parameter command will allow you to input the plot data of');
  MoveTo(10, 50);
  DrawString('the two parameter root locus method. The default values are shown');
```

```
 MoveTo(10,65);
 DrawString('in the dialog box. They can be changed as desired.');
  MoveTo(20, 80);
 DrawString('The items in this dialog are similar to those in the one parameter');
  MoveTo(10, 95);
 DrawString('plot data dialog box, but several items are different. ');
  MoveTo(20, 110);
 DrawString('There exists one more undetemined parameter B to be inserted.The');
  MoveTo(10, 125);
 DrawString('How many loci item lets you decide how many loci are to be drawn ');
  MoveTo(10, 140);
 DrawString('for each parameter. It will be 1 up to 10 for each parameter.');
  MoveTo(10, 155);
 DrawString('There is no auto scale for axes. Only the manual scale is available');
  MoveTo(10, 170);
 DrawString('The last item is the marking and justification in order to draw');
  MoveTo(10, 185);
 DrawString('the selected A and B values on the plot. Two buttons are chosen');
  MoveTo(10, 200);
 DrawString('each time for each parameter; one for position, the other justifi- ');
  MoveTo(10, 215);
  DrawString('cation to draw.');
  MoveTo(40, 260);
 DrawString('" CLICK THE MOUSE ONCE TO RETURN THE MAIN MENU "');
 repeat until Button;
  HideWindow(Window[HelpWind].P);
  SetPort(SavePort);
end;

procedure InfoPrint;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = }
{=          Bring up the Help information of print out.         = }
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = }
var
  SavePort : GrafPtr;
  InLeft    : integer; { Window offset from left }
  InTop     : integer; { Window offset from top  }

begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(HelpWind, FALSE);
  with Window[HelpWind].P^.portRect do
 begin
    InLeft := (XMaxGlb - (Right-Left)) div 2; { Calculate window offsets }
    InTop := (YMaxGlb - (Bottom-Top)) div 2;
 end;
  MoveWindow(Window[HelpWind].P, InLeft, InTop, TRUE); { Center the window }
```

```
  SetVisibility(HelpWind, TRUE);
  TextFont(SystemFont);
  TextSize(14);
  TextStyle([]);
  MoveTo(160, 15);
  DrawString('PRINT OUT');
  TextStyle([]);
  TextSize(12);
  MoveTo(20, 35);
  DrawString('In order to print out your work, there are a few ways available.');
  MoveTo(20, 50);
  DrawString('-.Select the print command in the File menu. This allows the ');
  MoveTo(10,65);
  DrawString('user to get a hard copy of any plot displayed.');
  MoveTo(20, 80);
  DrawString('-. Hold the command key and then type the number 4 to print ');
  MoveTo(10, 95);
  DrawString('the contents of the active window immediately.');
  MoveTo(20, 110);
  DrawString('-. Press the command and shift key and type the number 3 to ');
  MoveTo(10, 125);
  DrawString('create a MacPaint document.');
  MoveTo(40, 260);
  DrawString('" CLICK THE MOUSE ONCE TO RETURN THE MAIN MENU "');
  repeat until Button;
  HideWindow(Window[HelpWind].P);
  SetPort(SavePort);
end;

begin
end.{unit message}
```

```
unit MyDialog (7000);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=          This unit supports four dialog boxes for input data.        =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

{$B+}                { Set the bundle bit }
{$R MacRootLocus.Rsrc} { Identify resource file for menu and icon info }
{$T APPLFFTD}          { Set the application type and creator }
{$S+}               { generate segmented code}
{$I-}                { Turn off I/O  error checking }
{$U SpecVar}
{$U RootsFinder}
{$U MakeRoot}
{$U TurboGraph}
{$U Message}

interface
uses
    MemTypes, QuickDraw, OSIntf, ToolIntf,PackIntf,
    PasPrinter,SANE,MacPrint, RootsFinder, SpecVar,
  {$S Second Segment}
   TurboGraph,
  Message,
   {$S Third Segment}
  MakeRoot;

procedure MakeRegend;
procedure GetEQParameter;
procedure GetCoeff;
procedure PlotOneParameter;
procedure PLotTwoParameter;

implementation

procedure MakeRegend;
{= = = = = = = = = = = = = = =  = = = = = = = = = = = = = = = = =}
{=          This procedure provides the informaton box with          =}
{=         the capabilety of the text editor                          =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = }
var
  SavePort : GrafPtr;
  InLeft   : integer; { Window offset from left }
  InTop    : integer; { Window offset from top  }
 begin
  GetPort(SavePort); { save the current grafport }
  SelectWind(3, FALSE);
```

```
Textsize(9);
with Window[3].P^.portRect do
begin
  InLeft := Round((XMaxGlb - (Right-Left)) / 1.38); { Calculate window offsets }
  InTop := Round((YMaxGlb - (Bottom-Top)) /1.05);
end;
MoveWindow(Window[3].P, InLeft, InTop, TRUE); { Center the window }
SetVisibility(3, TRUE);
txRect := thePort^.portRect;
textH := TENew(txRect, txRect);
TEIdle(textH);
TextInputEnabled := True;
end;

procedure GetEQParameter;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=          This procedure provides the dialog box for the                =}
{=          equation parameter of the characteristic equation.            =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
const
 CancelBtn      = 1;
 OKBtn          = 2;
 InitDegreeText = 3;
 InitGuessText  = 4;
 MaxIterText    = 5;
 ToleranceText  = 6;
var
 tempdegree,tempIter : Longint;
 tempGuess : TNcomplex;
 tempTolerance : Extended;
 BoxPlot : Boolean;
begin
  theDialog := GetNewDialog(256, nil, pointer(-1));
  GetDItem(theDialog, 3, theType, h3, r);
  GetDItem(theDialog, 4, theType, h4, r);
  GetDItem(theDialog, 5, theType, h5, r);
  GetDItem(theDialog, 6, theType, h6, r);
 Done := False;
 tempGuess.Re := 1.0;
 tempGuess.Im := 0.0;
 tempIter := 100;
 tempTolerance := 1E-7;
 BoxPlot := False;
 repeat
   ModalDialog(nil, itemHit);
  case itemHit of
   CancelBtn     : done := True;
```

```
   OKBtn         : begin
                   InitDegree := tempdegree;
                  InitGuess.Re := tempGuess.Re;
                   InitGuess.Im := tempGuess.Im;
                     MaxIter := tempIter;
                   Tolerance := tempTolerance;
                  done := True;
                 end;
   InitDegreeText  : begin
                     GetIText(h3, s);
                     StringToNum(s, tempdegree);
                     InitDegreeStatus := True;
                      if (tempdegree <1 ) or (tempdegree >10) then
                     begin
                       sysbeep(10);
                     end;
                 end;
   InitGuessText  : begin
                     GetIText(h4, s);
                     tempGuess.Re := Str2Num(s);
                    if (tempGuess.Re <0 )  then
                     begin
                       sysbeep(10);
                     end;
                 end;
   MaxIterText  : begin
                     GetIText(h5, s);
                      StringToNum(s, tempIter);
                     if (tempIter <0 )  then
                     begin
                       sysbeep(10);
                     end;
                 end;
   ToleranceText  : begin
                     GetIText(h6, s);
                      tempTolerance := Str2Num(s);
                     if (tempTolerance <= 0 )  then
                     begin
                       sysbeep(10);
                     end;
                 end;
  end;
  until done;
   DisposDialog(theDialog);
 end;

procedure GetCoeff;
```

```
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=         This procedure provides the dialog box for the         =}
{=          coefficients of the characteristic equation.          =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
 type char_set = set of char;
 const
  CancelBtn      = 1;
  OKBtn          = 2;
  S0Text         = 3;
  S1Text         = 4;
  S2Text         = 5;
  S3Text         = 6;
  S4Text         = 7;
  S5Text         = 8;
  S6Text         = 9;
  S7Text         = 10;
  S8Text         = 11;
  S9Text         = 12;
  S10Text        = 13;

 var
   ldno,tempitemhit,sssindex,ssslength
          : Integer;
  h        : Array[1..13] of Handle;
   set_valid :char_set;
  op_set : char_set;
   sss: str255;
  sssc: char;
   noerrorcheck,leftparen: boolean;

   Procedure skipblank ;
  begin
     while ((sss[sssindex] = ' ') and (sssindex <= ssslength)) do
       sssindex := sssindex + 1;
  end;

   Procedure Checknext ( setofnext1,setofnext2:char_set);
   var opchar: char;
  begin
    opchar := sssc;
    sssindex := sssindex +1;
    sssc := sss[sssindex];
    if (sssc = ' ') then begin
        skipblank;
        sssc := sss[sssindex];
        if sssindex <= ssslength then begin
            if not (sssc in setofnext1) then
```

```
            noerrorcheck:= False;
     end else
         if (opchar in (op_set + ['('])) then
            noerrorcheck := false
  end else
       if not (sssc in setofnext2) then
           noerrorcheck:= false
end;

 Procedure CheckExpression(matchparen:boolean);
begin
    leftparen:= true;
    while ((noerrorcheck and leftparen) and (sssindex < ssslength)) do
   case (sssc) of
           '0'..'9':  Checknext(op_set+['.',')'],op_set+['0'..'9','.',')']);
         'a','b': Checknext(op_set+  [')'],op_set+['a','b',')']);
              '*','^','/','-','+':  Checknext(['0'..'9','a','b','('],['0'..'9','a','b','(']);
         '.': Checknext(['0'..'9'],['0'..'9']);
     '(':begin
                     Checknext(['0'..'9','a','b','-','('],['0'..'9','a','b','-','(']);
           CheckExpression (false);
           if sssc = ')' then
            if sssindex < ssslength then begin
               sssc := sss[sssindex];
                 if sss[sssindex + 1] = ' ' then skipblank;
              if sssindex < ssslength then
                        Checknext(op_set+[')'],op_set+[')'])
                else if not matchparen then noerrorcheck := false
           end else begin
                  if not matchparen then noerrorcheck := false
              end
           else noerrorcheck := false;
       end;
     ')':begin
           if matchparen then noerrorcheck:= false;
          leftparen := false;
       end;
     ' ':begin
          skipblank ;
          if sssindex = ssslength then begin
                if not (sss[sssindex] in ['0'..'9','a','b']) then
               noerrorcheck:= false
          end else if sssindex < ssslength then sssc := sss[sssindex];
      end
  end;

    if not noerrorcheck then sysbeep (30);
```

```
 end;
{begin
 if not InitDegreeStatus then
  begin
   SysBeep(30);
   AlertBox1;
  end
 else}
  begin
     set_valid  :=  ['0'..'9','a','b','+','*','/','-','  ','^','(',')','.'];
    op_set  :=  ['+','/','-','*','^'];
  if not InitDegreeStatus then
   begin
    SysBeep(30);
    AlertBox1;
   end
  else
   begin
    Idno := 300 + InitDegree;
     theDialog := GetNewDialog(Idno, nil, Pointer (-1));
    for n := 3 to InitDegree + 3 do
     GetDItem(theDialog, n, theType, h[n], r);
   done := False;
     ModalDialog(nil, itemHit);

   repeat

     tempitemhit := itemHit;
    case itemHit of
    CancelBtn      : done := True;
    OKBtn          : done := True;

    3..12          : begin
                      while tempitemhit = itemHit do
                          ModalDialog(nil, tempitemhit);
                       GetIText (h[itemHit],sss);
                     noerrorcheck := true;
                    sssindex:= 1;
                     ssslength := Length (sss);
                     sssc := sss[1];
                           if not (sss[1] in ['0'..'9','a','b','-','(']) then
                       noerrorcheck := False;
                      CheckExpression ( True);
                           GetIText(h[itemHit],InfixArray[InitDegree + 3 - itemHit]);
                   GetCoeffStatus := True;
                      itemhit := tempitemhit;
                     if itemhit = OKBtn then done := true
```

```
                    end;
     end;
     until done;
      DisposDialog(theDialog);
    end;
  end;
{end;}

procedure PlotOneParameter;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=              This procedure provides the dialog box for the          =}
{=             plot data of the one parameter root locus method.        =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
 const
  CancelBtn        = 1;
  OKBtn            = 2;
  LinearBtn        = 3;
   LogarithmicBtn    = 4;
  AutoBtn          = 5;
  ManualBtn        = 6;
  AMinGainText     = 7;
  AMaxGainText     = 8;
  XMinText         = 9;
  XMaxText         = 10;
  YMinText         = 11;
  YMaxText         = 12;
   PlotPointsText   = 13;

 var
  saveSoundVol : Integer;
   radButton   : array [3..6] of ControlHandle;
   h           : Array[7..13] of Handle;
  tempAMinGain   : Extended;
  tempAMaxGain   : Extended;
  tempXMin       : Extended;
  tempXMax       : Extended;
  tempYMin       : Extended;
  tempYMax       : Extended;
   tempPoints : LongInt;
  OkPlot        :boolean;
begin
 if not InitDegreeStatus or not GetCoeffStatus then
  begin
    SysBeep(30);
    AlertBox2;
  end
 else
```

```
begin
 GetSoundVol (saveSoundVol);
  SetSoundVol (1);
   theDialog := GetNewDialog(257, nil, Pointer (-1));
  for n := LinearBtn to ManualBtn do
     GetDItem(theDialog,n,theType,Handle(radButton[n]),r);
    SetCtlValue (radButton[LinearBtn],1);
  linear := True;
   SetCtlValue (radButton[AutoBtn],1);
 AutoScale := True;
 for n := 7 to 13 do
   GetDItem(theDialog, n, theType, h[n], r);
  tempAMinGain := Str2Num('0.1');
   tempAMaxGain := Str2Num('10000');
    tempXMin := Str2Num('-10');
   tempXMax := Str2Num('5');
    tempYMin := Str2Num('-10');
  tempYMax := Str2Num('10');
   StringToNum('50', tempPoints);
 done := False;
 OkPlot := False;
 repeat
    ModalDialog(nil,itemHit);
  case itemHit of
   CancelBtn       : done := True;
   OKBtn          : begin
                   done := True;
                    OkPlot := True;
                  end;
    LinearBtn       : begin
                    SetSoundVol (1);
                      for n := LinearBtn to LogarithmicBtn do
                          SetCtlValue(radButton[n],Ord(n = itemHit));
                     linear := True;
                  end;

    LogarithmicBtn   : begin
                    SetSoundVol (1);
                      for n := LinearBtn to LogarithmicBtn do
                          SetCtlValue(radButton[n],Ord(n = itemHit));
                     linear := False;
                  end;
   AutoBtn        : begin
                    SetSoundVol (1);
                    for n := AutoBtn to ManualBtn do
                          SetCtlValue(radButton[n],Ord(n = itemHit));
                   AutoScale := True;
```

```
              end;

ManualBtn        : begin
               SetSoundVol (1);
               for n := AutoBtn to ManualBtn do
                   SetCtlValue(radButton[n],Ord(n = itemHit));
              AutoScale := False;
              end;
AMinGainText     : begin
                 GetIText(h[itemHit] , s);
                tempAMinGain := Str2Num(s);
                if (tempAMinGain < 0 ) or  (tempAMinGain > 1e7) then
               begin
                 sysbeep(10);
               end;
              end;
AMaxGainText     : begin
                 GetIText(h[itemHit] , s);
                tempAMaxGain := Str2Num(s);
                if (tempAMaxGain < 0 ) or  (tempAMaxGain > 1e7) then
                begin
                  sysbeep(10);
                end;
              end;
XMinText         : begin
                 GetIText(h[itemHit] , s);
                tempXMin := Str2Num(s);
                 if (tempXMin <-1e7 ) or  (tempXMin > 1e7) then
               begin
                 sysbeep(10);
               end;
              end;
XMaxText         : begin
                 GetIText(h[itemHit] , s);
                tempXMax := Str2Num(s);
                if (tempXMax <-1e7 ) or  (tempXMax > 1e7) then
               begin
                 sysbeep(10);
               end;
              end;
YMinText         : begin
                 GetIText(h[itemHit] , s);
                tempYMin := Str2Num(s);
                 if (tempYMin <-1e7 ) or  (tempYMin > 1e7) then
               begin
                 sysbeep(10);
               end;
```

```
                  end;
    YMaxText          : begin
                       GetIText(h[itemHit] , s);
                      tempYMax := Str2Num(s);
                       if (tempYMax <-1e7 ) or  (tempYMax > 1e7) then
                     begin
                       sysbeep(10);
                     end;
                  end;
    PlotPointsText   : begin
                       GetIText(h[itemHit] , s);
                      StringToNum(s, tempPoints);
                       if (tempPoints <= 0) or  (tempPoints > 150) then
                     begin
                       sysbeep(10);
                     end;
                  end;

   end;{case end}
    until done;
     DisposDialog(theDialog);
    if OkPlot then
    begin
     AMinGain  := tempAMinGain;
     AMaxGain  := tempAMaxGain;
     XMn       := tempXMin;
     XMx       := tempXMax;
     YMn       := tempYMin;
     YMx       := tempYMax;
      Points    := tempPoints;
      HideCursor;
       Rewrite(OutFile,'RootLocus1.data');
     GetRoot1;
      Close(OutFile);
      ShowCursor;
     MakeRegend;
      end;{if}
  end;{case}
end;{PlotOneParameter}

procedure  PLotTwoParameter;
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=            This procedure provides the dialog box for the              =}
{=           plot data of the two parameter root locus method.          =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
 const
  CancelBtn        = 1;
```

```
OKBtn          = 2;
LinearBtn        = 3;
LogarithmicBtn    = 4;
AMarkStBtn       = 5;
AMarkEdBtn       = 6;
AJustifyRBtn     = 7;
AJustifyLBtn     = 8;
BMarkStBtn       = 9;
BMarkEdBtn       = 10;
BJustifyRBtn     = 11;
BJustifyLBtn     = 12;
AMinGainText     = 13;
AMaxGainText     = 14;
BMinGainText     = 15;
BManGainText     = 16;
LociText         = 17;
PlotPointsText   = 18;
XMinText         = 19;
XMaxText         = 20;
YMinText         = 21;
YMaxText         = 22;

var
 saveSoundVol,i : Integer;
 radButton   : array [3..12] of ControlHandle;
 h          : Array[13..22] of Handle;
 tempAMinGain   : Extended;
 tempAMaxGain   : Extended;
 tempBMinGain   : Extended;
 tempBMaxGain   : Extended;
 tempXMin       : Extended;
 tempXMax       : Extended;
 tempYMin       : Extended;
 tempYMax       : Extended;
 tempStep, tempPoints : LongInt;
 OkPlot : Boolean;
begin
 if not InitDegreeStatus or not GetCoeffStatus then
  begin
   SysBeep(30);
   AlertBox2;
  end
 else
  begin
   GetSoundVol (saveSoundVol);
   SetSoundVol (1);
    theDialog := GetNewDialog(258, nil, Pointer (-1));
```

```
for n := LinearBtn to BJustifyLBtn do
   GetDItem(theDialog,n,theType,Handle(radButton[n]),r);
 SetCtlValue (radButton[LinearBtn],1);
linear := True;
 SetCtlValue (radButton[AMarkStBtn],1);
AMarkStatus := False;
 SetCtlValue (radButton[AJustifyRBtn],1);
ARJustification := True;
 SetCtlValue (radButton[BMarkStBtn],1);
BMarkStatus := False;
 SetCtlValue (radButton[BJustifyRBtn],1);
BRJustification := True;
for n := 13 to 22 do
  GetDItem(theDialog, n, theType, h[n], r);
tempAMinGain := Str2Num('0.1');
 tempAMaxGain := Str2Num('10000');
tempBMinGain := Str2Num('0.1');
 tempBMaxGain := Str2Num('10000');
 tempXMin := Str2Num('-10');
 tempXMax := Str2Num('5');
 tempYMin := Str2Num('-10');
tempYMax := Str2Num('10');
 StringToNum('4', tempStep);
 StringToNum('50', tempPoints);
done := False;
OkPlot := False;
repeat
   ModalDialog(nil,itemHit);
 case itemHit of
  CancelBtn    : done := True;
  OKBtn        : begin
                 done := True;
                  OkPlot := True;
                end;
   LinearBtn      : begin
                     SetSoundVol (1);
                      for n := LinearBtn to LogarithmicBtn do
                          SetCtlValue(radButton[n],Ord(n = itemHit));
                      linear := True;
                  end;

   LogarithmicBtn  : begin
                     SetSoundVol (1);
                      for n := LinearBtn to LogarithmicBtn do
                          SetCtlValue(radButton[n],Ord(n = itemHit));
                      linear := False;
                   end;
```

```
AMarkStBtn      : begin
            SetSoundVol (1);
            for n := AMarkStBtn to AMarkEdBtn do
                SetCtlValue(radButton[n],Ord(n = itemHit));
            AMarkStatus := False;
          end;

AMarkEdBtn     : begin
            SetSoundVol (1);
            for n := AMarkStBtn to AMarkEdBtn do
                SetCtlValue(radButton[n],Ord(n = itemHit));
            AMarkStatus := True;
          end;
AJustifyRBtn    : begin
            SetSoundVol (1);
             for n := AJustifyRBtn to AJustifyLBtn do
                SetCtlValue(radButton[n],Ord(n = itemHit));
             ARJustification := True;
          end;

AJustifyLBtn    : begin
            SetSoundVol (1);
             for n := AJustifyRBtn to AJustifyLBtn do
                SetCtlValue(radButton[n],Ord(n = itemHit));
             ARJustification := False;
          end;
BMarkStBtn      : begin
            SetSoundVol (1);
            for n := BMarkStBtn to BMarkEdBtn do
                SetCtlValue(radButton[n],Ord(n = itemHit));
            BMarkStatus := False;
          end;

BMarkEdBtn     : begin
            SetSoundVol (1);
            for n := BMarkStBtn to BMarkEdBtn do
                SetCtlValue(radButton[n],Ord(n = itemHit));
            BMarkStatus := True;
          end;
BJustifyRBtn    : begin
            SetSoundVol (1);
             for n := BJustifyRBtn to BJustifyLBtn do
                SetCtlValue(radButton[n],Ord(n = itemHit));
             BRJustification := True;
          end;

BJustifyLBtn    : begin
```

```
            SetSoundVol (1);
              for n := BJustifyRBtn to BJustifyLBtn do
                 SetCtlValue(radButton[n],Ord(n = itemHit));
              BRJustification := False;
            end;
AMinGainText   : begin
              GetIText(h[itemHit],s);
             tempAMinGain := Str2Num(s);
             if (tempAMinGain < 0 ) or  (tempAMinGain > 1e7) then
            begin
              sysbeep(10);
            end;
          end;
AMaxGainText   : begin
              GetIText(h[itemHit],s);
             tempAMaxGain := Str2Num(s);
             if (tempAMaxGain < 0 ) or  (tempAMaxGain > 1e7) then
            begin
              sysbeep(10);
            end;
          end;
BMinGainText   : begin
              GetIText(h[itemHit],s);
             tempBMinGain := Str2Num(s);
             if (tempBMinGain < 0 ) or  (tempBMinGain > 1e7) then
            begin
              sysbeep(10);
            end;
          end;

BManGainText   : begin
              GetIText(h[itemHit],s);
             tempBMaxGain := Str2Num(s);
             if (tempBMaxGain < 0 ) or  (tempBMaxGain > 1e7) then
            begin
              sysbeep(10);
            end;
          end;
XMinText       : begin
              GetIText(h[itemHit],s);
             tempXMin := Str2Num(s);
              if (tempXMin <-1e7 ) or  (tempXMin > 1e7) then
            begin
              sysbeep(10);
            end;
          end;
XMaxText       : begin
```

```
                GetIText(h[itemHit],s);
               tempXMax := Str2Num(s);
                if (tempXMax <-1e7 ) or  (tempXMax > 1e7) then
              begin
                sysbeep(10);
              end;
            end;
  YMinText      : begin
                GetIText(h[itemHIt],s);
               tempYMin := Str2Num(s);
                if (tempYMin <-1e7 ) or  (tempYMin > 1e7) then
              begin
                sysbeep(10);
              end;
            end;
  YMaxText      : begin
                GetIText(h[itemHit],s);
               tempYMax := Str2Num(s);
                if (tempYMax <-1e7 ) or  (tempYMax > 1e7) then
              begin
                sysbeep(10);
              end;
            end;
  LociText     : begin
                GetIText(h[itemHit],s);
               StringToNum(s, tempStep);
                if (tempStep <= 0) or  (tempStep > 15) then
              begin
                sysbeep(10);
              end;
            end;
  PlotPointsText :  begin
                GetIText(h[itemHit],s);
               StringToNum(s, tempPoints);
                if (tempPoints <= 0) or  (tempPoints > 150) then
              begin
                sysbeep(10);
              end;
            end;
end;{case end}
 until done;
  DisposDialog(theDialog);
 if OkPlot then
 begin
  AMinGain   := tempAMinGain;
  AMaxGain   := tempAMaxGain;
  BMinGain   := tempBMinGain;
```

```
    BMaxGain   := tempBMaxGain;
    XMn        := tempXMin;
    XMx        := tempXMax;
    YMn        := tempYMin;
    YMx        := tempYMax;
    Step       := tempStep;
     Points     := tempPoints;
     HideCursor;
      Rewrite(OutFile,'RootLocus2.data');
    GetRoot2;
     Close(OutFile);
     ShowCursor;
    MakeRegend;
     end;{if}
  end;{case}
end;{PlotTwoParameter}

begin
end.{unit  MyDialog}
```

```
unit  TurboGraph(11000);

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}
{=                                                                                     =}
{=      Turbo Pascal Numerical Methods Toolbox                                         =}
{=       Copyright (C) 1987 Borland International                                      =}
{=                                                                                     =}
{=       This unit provides routines for displaying graphics.                          =}
{=                                                                                     =}
{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =}

{$U RootsFinder}
{$U SpecVar}

interface

uses
   MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,PasPrinter, SANE, MacPrint ,
  RootsFinder, SpecVar;

const
  MaxWorldsGlb   = 10;  { The maximum number of worlds that can be defined }
  MaxWindowsGlb  = 10;  { The maximum number of windows that can be defined }
  {MaxPlotGlb      = 1500;}{ The maximum number of points in a plot array }

  X1Offset       = 40;  { The upper left corner of the axis }
  Y1Offset       = 10;
  X2Offset       = 30;  { The lower right corner of the axis }
  Y2Offset       = 90;

type
  WrkString  = Str255; { A general string type }

  WorldType  = record  { Used to store world coordinates }
             X1, Y1, X2, Y2 : real;
           end;

  WindowType = record  { Used to store window information }
              X1, Y1, X2, Y2 : integer; { The windows screen coordinates }
              Header : WrkString;       { Header for a window }
             W      : WindowRecord;   { Mac window record }
              P      : WindowPtr;       { Mac window pointer }
             H      : PicHandle;       { A handle to a picture }
           end;

  Worlds      = array[1..MaxWorldsGlb] of WorldType; { Holds world info }
```

```
 Windows     = array[1..MaxWindowsGlb] of WindowType;{ Holds window info }
{ PlotArray  = array[1..MaxPlotGlb, 1..2] of Extended;}

var
 { Coordinates of the currently active window }
 XMinGlb, YMinGlb, XMaxGlb, YMaxGlb : integer;

 { Coordinates of the currently selected world }
 X1WldGlb, X2WldGlb, Y1WldGlb, Y2WldGlb : real;

 { World coordinate scaling factors }
 AxGlb, AyGlb, BxGlb, ByGlb : real;

 { Coordinates of an axis if one is defined }
 X1RefGlb, X2RefGlb, Y1RefGlb, Y2RefGlb : integer;

 { The maximum world and window number defined }
 MaxWorldGlb, MaxWindowGlb : integer;

 { The currently selected world and window }
 WorldNdxGlb, WindowNdxGlb : integer;

 { Aspect ratio for a true circle }
 AspectFactor : real;
 AspectGlb    : real;

 { Flags if an axis is defined }
 AxisGlb : boolean;

 { Flags if hatching is turned on }
 HatchGlb : boolean;

 { Flags if clipping should be performed }
 ClippingGlb : boolean;

 { The currently selected line style }
 LineStyleGlb : integer;

 { The current foreground color }
 ForeColorGlb : integer;

 { Holds the worlds defined by the user }
 World : Worlds;

 { Holds the windows defined by the user }
 Window : Windows;
```

```
procedure Error(ProcName : WrkString);
{ Report that an error occurred in the procedure named "ProcName" }

procedure SetForegroundColor(Color : integer);
{ Set the foreground drawing color }

procedure SetBackgroundColor(Color : integer);
{ Set the background color }

procedure DP(X, Y : integer);
{ Plot a pixel at position (X, Y) }

function PD(X, Y : integer) : boolean;
{ Return true if the color of the pixel at (X, Y) matches ForeColorGlb }

procedure DrawStraight(X1, X2, Y : integer);
{ Draw a horizontal line from X1,Y to X2,Y }

procedure InvertWindow(WinNum : integer);
{ Invert the window referenced by WinNum }

function RealToStr(R : real) : WrkString;
{ Returns the string representation of the real R. }

function IntToStr(I : integer) : WrkString;
{ Returns the string representation of the integer I. }

procedure SetLineStyle(Style : integer);
{ Select the current line style }

function GetLineStyle : integer;
{ Returns the current line style }

procedure DefineWorld(WorldNum : integer; X_1, Y_1, X_2, Y_2 : real);
{ Defines a world coordinate system with a specific number }

procedure SelectWorld(WorldNum : integer);
{ Select the world associated with WorldNum }

procedure DefineWindow(WinNum, XLo, YLo, XHi, YHi, WindowType : integer);
{ Defines a window with a specific window number and window type }

procedure SetVisibility(WinNum : integer; Visible : boolean);
{ Sets the visibility of a window to either TRUE or FALSE }

procedure SelectWind(WinNum : integer; Visible : boolean);
{ Selects a window as visible or invisible }
```

```
procedure DefineHeader(WinNum : integer; Hdr : WrkString);
{ Define a header for a window }

procedure RemoveHeader(WinNum : integer);
{ Clears a header from a window }

procedure ReDefineWindow(WinNum, X1, Y1, X2, Y2, WindowType : integer);
{ Redefines the coordinates of an existing window }

function WindowX(X : real) : integer;
{ Converts an X world coordinate into a screen coordinate }

function WindowY(Y : real) : integer;
{ Converts a Y world coordinate into a screen coordinate }

procedure ResetWorlds;
{ Resets all worlds to the maximum dimensions of the screen }

procedure InitGraphic;
{ Initializes the graphics system }

function Clip(var X1, Y1, X2, Y2 : integer) : boolean;
{ Clips a line to the current window and returns TRUE if any part }
{ of the line is still in the window after the clip operation      }

procedure DrawPoint(Xr, Yr : real);
{ Draw a point in world coordinates }

function PointDrawn(Xr, Yr : real) : boolean;
{ Returns TRUE if the point at (Xr, Yr) is drawn }

procedure DrawLine(X1, Y1, X2, Y2 : real);
{ Draw a line in world coordinates }

procedure DrawLineClipped (X1, Y1, X2, Y2 : integer);
{ Draw a line clipped in screen coordinates }

procedure DrawCrossDiag(X, Y, Scale : integer);
{ Draw a cross at screen coordinate (X, Y) with a scaling factor }

procedure DrawWye(X, Y, Scale : integer);
{ Draw a Y at screen coordinate (X, Y) with a scaling factor }

procedure DrawDiamond(X, Y, Scale : integer);
{ Draw a cross at screen coordinate (X, Y) with a scaling factor }
```

```
procedure DrawCircleDirect(Xr, Yr, R : integer);
{ Draw a circle at screen coordinate (Xr, Yr) with a radius R }

procedure DrawCircle(X_R, Y_R, Xradius : real);
{ Draw a circle at world coordinate (X_R, Y_R) with a radius R }

procedure DrawCross(X1, Y1, Scale : integer);
{ Draw a cross at screen coordinate (X, Y) with a scaling factor }

procedure DrawStar(X, Y, Scale : integer);
{ Draw a star at screen coordinate (X, Y) with a scaling factor }

procedure DrawSquareC(X1, Y1, X2, Y2 : integer; Fill : boolean);
{ Draw a square in screen coordinates with optional filling }

procedure DrawSquare(X1, Y1, X2, Y2 : real; Fill : boolean);
{ Draw a square in world coordinates with optional filling }

procedure DrawAscii(X, Y : integer; Size, CharByte : byte);
{ Draw a character with ASCII code CharByte }

procedure DrawText(X, Y, Scale : integer; Txt : Str255);
{ Draw a string at screen coordinate (X, Y) with a scaling factor }

procedure DrawTextW(X, Y : real; Scale : integer; Txt : WrkString);
{ Draw a string at world coordinate (X, Y) with a scaling factor }

procedure TextStyle(Face : Style);
{ Face = (bold, italic, underline, outline, shadow, condense, extend) }

procedure HardCopy(TopWin : boolean);
{ Do a screen dump of the currently selected window (TopWin = TRUE) }
{ or the whole screen (TopWin = FALSE) }

procedure OpenPic(WinNum : integer; ShowPic : boolean);
{ Open a picture for a specific window and only show the drawing }
{ if ShowPic is set to TRUE }

procedure DrawPic(WinNum : integer);
{ Draw the picture associated with a window }

procedure ErasePic(WinNum : integer);
{ Erase the picture associated with a window }

procedure ClearWindow(WinNum : integer);
{ Clear the content portion of a window }
```

```
function WhereX : integer;
{ Returns the X cursor position }

function WhereY : integer;
{ Returns the Y cursor position }

procedure SetWindow(X1, Y1, X2, Y2 : integer);
{ Creates an invisible window inset from the currently selected one }

procedure FindWorld(I : integer; A : PlotArray; NPoints : integer);
{ Finds a world to fit a polygon defined in A }

procedure FindWorld1(I : integer; XMn,YMn,XMx,YMx:ExTended );
{ Finds a world to fit a polygon defined in A }

procedure DrawAxis(Footer1, Footer2 : WrkString; Arrows : boolean);
{ Draws an axis with Footers and optional arrows on the axis }

procedure ResetAxis;
{ Sets AxisGlb to TRUE }

procedure DrawPolygon(A : PlotArray; First, NPoints, Line, Scale,
                 Lines : integer; CrossHairs : boolean);

{ Draws a polygon defined in A with "NPoints" points, line style "Line" }
{ and optional Lines from the axis to the points and cross hairs.      }

procedure Hatch(X_1, Y_1, X_2, Y_2, Delta : real);
{ Hatch a bar in a histogram }

procedure DrawHistogram(A :PlotArray; NPoints : integer;
                  Hatching : boolean; HatchStyle : integer);
{ Draw a histogram defined in A with "NPoints" points and optional }
{ hatching with a given hatch style }

implementation

procedure Error{(ProcName : WrkString)};
{ Report that an error occurred in the procedure named "ProcName" }
begin
  DrawString('ERROR in ');
  DrawString(ProcName);
  DrawString(' Press the Button to exit.');
  repeat until Button;
  Halt;
end; { Error }
```

```
procedure SetForegroundColor{(Color : integer)};
{ Set the foreground drawing color }
begin
 case Color of
   0 : begin
          ForeColor(BlackColor);
         ForeColorGlb := 0;
      end;
   1 : begin
          ForeColor(WhiteColor);
         ForeColorGlb := 1;
      end;
 end;
end; { SetForegroundColor }

procedure SetBackgroundColor{(Color : integer)};
{ Set the background color }
begin
 case Color of
    0 : BackColor(BlackColor);
    1 : BackColor(WhiteColor);
 end;
end; { SetBackgroundColor }

procedure DP{(X, Y : integer)};
{ Plot a pixel at position (X, Y) }
begin
  MoveTo(X, Y);
  LineTo(X, Y);
end; { DP }

function PD{(X, Y : integer) : boolean};
{ Return true if the color of the pixel at (X, Y) matches ForeColorGlb }
var
  BlackPixel : boolean;
  PixelColor : integer;
begin
   BlackPixel := GetPixel(X, Y);
  if BlackPixel then
    PixelColor := 0
 else
    PixelColor := 1;
  PD := PixelColor = ForeColorGlb;
end; { PD }

procedure DrawStraight{(X1, X2, Y : integer)};
{ Draw a horizontal line from X1,Y to X2,Y }
```

```
begin
  MoveTo(X1, Y);
  LineTo(X2, Y);
end; { DrawStraight }

procedure InvertWindow{(WinNum : integer)};
{ Invert the window referenced by WinNum }
begin
  if WinNum in [1..MaxWindowGlb] then
     InvertRect(Window[WinNum].W.Port.PortRect)
  else
     Error('Invert Window');
end; { InvertWindow }

function RealToStr{(R : real) : WrkString};
{ Returns the string representation of the real R. }
var
  Int, Frac : LongInt;
  S1, S2 : Str255;
  Negitive : boolean;
begin
  S1 := '';
  S2 := '';
  if R < 0.0 then
   Negitive := TRUE
  else
   Negitive := FALSE;
  R := ABS(R);
  Int := Trunc(R);
  Frac := Round(100.0 * (R - Int));
  NumToString(Int, S1);
  NumToString(Frac, S2);
  if Length(S2) = 1 then
    S2 := S2 + '0';
  S2 := S1 + '.' + S2;
  if Negitive then
    RealToStr := '-' + S2
  else
    RealToStr := ' ' + S2;
end; { RealToStr }

function IntToStr{(I : integer) : WrkString};
{ Returns the string representation of the integer I. }
var
  Form : DecForm;
  Str  : DecStr;
begin
```

```
   Form.Style := FixedDecimal;
   Form.Digits := 0;
    Num2Str(Form, I, Str);
   IntToStr := Str;
end; { IntToStr }

procedure SetLineStyle{(Style : integer)};
var
  LineStyle : Pattern;
begin
 case Style of
    0 : LineStyle := Black;
    1 : LineStyle := White;
    2 : LineStyle := Gray;
    3 : LineStyle := LtGray;
    4 : LineStyle := DkGray;
  otherwise
    Style := 0;
    LineStyle := Black;
 end;
  LineStyleGlb := Style;
   PenPat(LineStyle);
end; { SetLineStyle }

function GetLineStyle{ : integer};
begin
  GetLinestyle := LineStyleGlb;
end; { GetLineStyle }

procedure DefineWorld{(WorldNum : integer; X_1, Y_1, X_2, Y_2 : real)};
begin
  if ((X_1 <> X_2) and (Y_1 <> Y_2)) and
      (WorldNum in [1..MaxWorldsGlb]) then
     with World[WorldNum] do
   begin
     X1 := X_1;
     Y1 := Y_1;
     X2 := X_2;
     Y2 := Y_2;
     if WorldNum > MaxWorldGlb then
       MaxWorldGlb := WorldNum;
   end
  else if WorldNum in [1..MaxWorldsGlb] then
      Error('DefineWorld #1')
 else
      Error('DefineWorld #2');
end; { DefineWorld }
```

```
procedure SelectWorld{(WorldNum : integer)};
begin
  if (WorldNum in [1..MaxWorldGlb]) then
    with World[WorldNum] do
   begin
     WorldNdxGlb := WorldNum;
     X1WldGlb := X1;
     Y1WldGlb := Y1;
     X2WldGlb := X2;
     Y2WldGlb := Y2;
  end
 else
     Error('SelectWorld');
end; { SelectWorld }

procedure DefineWindow{(WinNum, XLo, YLo, XHi, YHi, WindowType : integer; Visible :
boolean)};
var
 BoundsRect : Rect;
 Title       : Str255;
 RefCon     : LongInt;
 Visible    : boolean;
 GoAwayFlag : boolean;

begin
  if WinNum in [1..MaxWindowsGlb] then
   begin
     if WinNum > MaxWindowGlb then
       MaxWindowGlb := WinNum;
    with BoundsRect do
    begin
      Left := XLo;
      Top := YLo;
      Right := XHi;
      Bottom := YHi;
    end;
      Window[WinNum].X1 := XLo;
      Window[WinNum].Y1 := YLo;
      Window[WinNum].X2 := XHi;
      Window[WinNum].Y2 := YHi;
      Title := '';
    GoAwayFlag := TRUE;
     Visible := FALSE;
     RefCon := WinNum;
     if WindowType = documentProc then
       WindowType := 16 * WindowType + 8; { Add Zoom box }
```

```
      Window[WinNum].P := NewWindow(@Window[WinNum].W, BoundsRect,
                                   Title, Visible, WindowType,
                                  POINTER(-1), GoAwayFlag, RefCon);
   end
  else
     Error('DefineWindow');
end; { DefineWindow }

procedure SetVisibility{(WinNum : integer; Visible : boolean)};
begin
   ShowHide(Window[WinNum].P, Visible);
end; { SetVisibility }

procedure SelectWind{(WinNum : integer; Visible : boolean)};
begin
  if (WinNum in [1..MaxWindowGlb]) then
   begin
      SelectWindow(Window[WinNum].P);
     if Visible then
        ShowWindow(Window[WinNum].P);
      SetPort(@Window[WinNum].W.Port);
     with Window[WinNum] do
    begin
       WindowNdxGlb := WinNum;
        X1RefGlb := W.Port.PortRect.Left;
        Y1RefGlb := W.Port.PortRect.Top;
        X2RefGlb := W.Port.PortRect.Right;
        Y2RefGlb := W.Port.PortRect.Bottom;
         BxGlb := (X2RefGlb - X1RefGlb - 16) / (X2WldGlb - X1WldGlb);
         ByGlb := (Y2RefGlb - Y1RefGlb - 16) / (Y2WldGlb - Y1WldGlb);
       AxGlb := X1RefGlb - X1WldGlb * BxGlb;
       AyGlb := Y1RefGlb - Y1WldGlb * ByGlb;
      AxisGlb := FALSE;
    end;
   end
  else
     Error('SelectWind');
end; { SelectWind }

procedure DefineHeader{(WinNum : integer; Hdr : WrkString)};
begin
  Window[WinNum].Header := Hdr;
   SetWTitle(Window[WinNum].P, Hdr);
end; { DefineHeader }

procedure RemoveHeader{(WinNum : integer)};
begin
```

```
  DefineHeader(WinNum, '');
end; { RemoveHeader }

procedure ReDefineWindow{(WinNum, X1, Y1, X2, Y2, WindowType : integer)};
begin
  if (WinNum in [1..MaxWindowsGlb]) then
   begin
     CloseWindow(Window[WinNum].P);
     Window[WinNum].P := NIL;
     DefineWindow(WinNum, X1, Y1, X2, Y2, WindowType);
     SelectWind(WinNum, FALSE);
     DefineHeader(WinNum, Window[WinNum].Header);
   end
  else
     Error('ReDefineWindow');
end; { ReDefineWindow }

function WindowX{(X : real) : integer};
var
  Temp : real;
begin
  Temp := AxGlb + BxGlb * X;
  if Temp > MaxInt then
    WindowX := MaxInt
  else if Temp < -32767 then
     WindowX := -32767
  else
     WindowX := trunc(Temp);
end; { WindowX }

function WindowY{(Y : real) : integer};
var
  Temp : real;
begin
  Temp := AyGlb + ByGlb * Y;
  if Temp > MaxInt then
    WindowY := MaxInt
  else if Temp < -32767 then
     WindowY := -32767
  else
     WindowY := trunc(Temp);
end; { WindowY }

procedure ResetWorlds;
var
  I : integer;
begin
```

```
  for I := 1 to MaxWorldsGlb do
    DefineWorld(I, XMinGlb, YMinGlb, XMaxGlb, YMaxGlb);
   SelectWorld(1);
end; { ResetWorlds }

procedure InitGraphic;
var

  Index      : integer;
  BoundsRect : Rect;
  Title      : Str255;
  Visible    : boolean;
  RefCon     : LongInt;
  GoAwayFlag : boolean;
  WindowType : integer;
begin
  XMinGlb := screenBits.bounds.Left;
  YMinGlb := screenBits.bounds.Top;
  XMaxGlb := screenBits.bounds.Right;
  YMaxGlb := screenBits.bounds.Bottom;
  for Index := 1 to MaxWindowsGlb do
  begin
    Window[Index].P := NIL;
    Window[Index].H := NIL;
  end;
  ResetWorlds;
  MaxWorldGlb := 0;
  MaxWindowGlb := 0;
  WindowNdxGlb := 0;
  WorldNdxGlb := 0;
  AspectFactor := 0.44;
  AspectGlb := ABS(AspectFactor) * AspectFactor;
  AxisGlb := false;
  HatchGlb := false;
  ClippingGlb := true;
  SetLineStyle(0); { Solid Black lines }
end; { InitGraphic }

function Clip{(var X1, Y1, X2, Y2 : integer) : boolean};
var
  Ix1, Iy1, Ix2, Iy2, Dummy, X1Loc, X2Loc : integer;
  ClipLoc : boolean;
  Temp : real;

function Inside(X, Xx1, Xx2 : integer) : integer;
begin
  Inside := 0;
```

```
  if X < Xx1 then
    Inside := -1
  else if X > Xx2 then
    Inside := 1;
end; { Inside }

begin  { Clip }
  Clip := true;
  ClipLoc := true;
  if ClippingGlb then
   begin
     X1Loc := X1RefGlb;
     X2Loc := X2RefGlb;
      Ix1 := Inside(X1, X1Loc, X2Loc);
      Iy1 := Inside(Y1, Y1RefGlb, Y2RefGlb);
      Ix2 := Inside(X2, X1Loc, X2Loc);
      Iy2 := Inside(Y2, Y1RefGlb, Y2RefGlb);
      if (Ix1 or Ix2 or Iy1 or Iy2) <> 0 then
     begin
       if X1 <> X2 then
      begin
        if Ix1 <>0 then
       begin
         if Ix1 < 0 then
          Dummy := X1Loc
        else
          Dummy := X2Loc;
        if Y2 <> Y1 then
        begin
            Temp := (Y2 - Y1) / (X2 - X1) * (Dummy - X1);
          if Temp > MaxInt then
            Temp := MaxInt
           else if Temp < -32767 then
             Temp := -32767;
           Y1 := Y1 + trunc(Temp);
        end;
         X1 := Dummy;
       end;
        if (Ix2 <> 0) and (X1 <> X2) then
       begin
         if Ix2 < 0 then
          Dummy := X1Loc
        else
          Dummy := X2Loc;
        if Y2 <> Y1 then
        begin
            Temp := (Y2 - Y1) / (X2 - X1) * (Dummy - X1);
```

```
      if Temp > MaxInt then
        Temp := MaxInt
        else if Temp < -32767 then
          Temp := -32767;
        Y2 := Y1 + trunc(Temp);
    end;
     X2 := Dummy;
  end;
    Iy1 := Inside(Y1, Y1RefGlb, Y2RefGlb);
    Iy2 := Inside(Y2, Y1RefGlb, Y2RefGlb);
end;
if Y1 <> Y2 then
begin
   if Iy1 <> 0 then
  begin
     if Iy1 < 0 then
      Dummy := Y1RefGlb
    else
      Dummy := Y2RefGlb;
    if X1 <> X2 then
    begin
          Temp := (X2 - X1) / (Y2 - Y1) * (Dummy - Y1);
      if Temp > MaxInt then
        Temp := MaxInt
        else if Temp < -32767 then
          Temp := -32767;
        X1 := X1 + trunc(Temp);
    end;
     Y1 := Dummy;
  end;
   if Iy2 <> 0 then
  begin
     if Iy2 < 0 then
      Dummy := Y1RefGlb
    else
      Dummy := Y2RefGlb;
    if X1 <> X2 then
    begin
          Temp := (X2 - X1) / (Y2 - Y1) * (Dummy - Y1);
      if Temp > MaxInt then
        Temp := MaxInt
        else if Temp < -32767 then
          Temp := -32767;
        X2 := X1 + trunc(Temp);
    end;
     Y2 := Dummy;
  end;
```

```
      end;
        Iy1 := Inside(Y1, Y1RefGlb, Y2RefGlb);
        Iy2 := Inside(Y2, Y1RefGlb, Y2RefGlb);
        if (Iy1 <> 0) or (Iy2 <> 0) then
         ClipLoc := false;
       if ClipLoc then
      begin
          Ix1 := Inside(X1, X1Loc, X2Loc);
          Ix2 := Inside(X2, X1Loc, X2Loc);
          if (Ix2 <> 0) or (Ix1 <> 0) then
          ClipLoc := false;
      end;
       Clip := ClipLoc;
    end;
  end;
end; { Clip }

procedure DrawPoint{(Xr, Yr : real)};
var
 X, Y : integer;
begin
  X := WindowX(Xr);
  Y := WindowY(Yr);
  DP(X, Y);
end; { DrawPoint }

function PointDrawn{(Xr, Yr : real) : boolean};
begin
   PointDrawn := PD(WindowX(Xr), WindowY(Yr));
end; { PointDrawn }

procedure DrawLine{(X1, Y1, X2, Y2 : real)};
begin
   MoveTo(WindowX(X1), WindowY(Y1));
   LineTo(WindowX(X2), WindowY(Y2));
end; { DrawLine }

procedure DrawLineClipped{(X1, Y1, X2, Y2 : integer)};
begin
  if Clip(X1, Y1, X2, Y2) then
 begin
    MoveTo(X1, Y1);
    LineTo(X2, Y2);
{ end
 else
 begin
    Moveto(X1 - 3,Y1 - 3);
```

```
    TextSize(9);
     DrawString(Ds);
   LabelSet := False;}
  end;
end; { DrawLineClipped }

procedure DrawCrossDiag{(X, Y, Scale : integer)};
begin
  DrawLineClipped(X - Scale, Y + Scale, X + Scale + 1, Y - Scale - 1);
  DrawLineClipped(X - Scale, Y - Scale, X + Scale + 1, Y + Scale + 1);
end; { DrawCrossDiag }

procedure DrawWye{(X, Y, Scale : integer)};
begin
  DrawLineClipped(X - Scale, Y - Scale, X, Y);
  DrawLineClipped(X + Scale, Y - Scale, X, Y);
  DrawLineClipped(X, Y, X, Y + Scale);
end; { DrawWye }

procedure DrawDiamond{(X, Y, Scale : integer)};
begin
  DrawLineClipped(X - Scale, Y, X, Y - Scale - 1);
  DrawLineClipped(X, Y - Scale + 1, X + Scale, Y + 1);
  DrawLineClipped(X + Scale, Y + 1, X, Y + Scale);
  DrawLineClipped(X, Y + Scale, X - Scale, Y);
end; { DrawDiamond }

procedure DrawCircleDirect{(Xr, Yr, R : integer)};

type
   Circ = array[1..14] of integer;

var
  Xk1, Xk2, Yk1, Yk2, Xp1, Yp1, Xp2, Yp2 : integer;
  Xfact, Yfact : real;
  I : integer;
  X : Circ;

procedure InitX;
begin
  X[1] := 0;
  X[2] := 121;
  X[3] := 239;
  X[4] := 355;
  X[5] := 465;
  X[6] := 568;
  X[7] := 663;
```

```
  X[8] := 749;
  X[9] := 823;
  X[10] := 885;
  X[11] := 935;
  X[12] := 971;
  X[13] := 993;
  X[14] := 1000;
end; { InitX }

begin { DrawCircleDirect }
  InitX;
  Xfact := abs(R);
  Yfact := Xfact * AspectGlb;
  if Xfact > 0.0 then
   begin
      Xk1 := trunc(X[1] * Xfact + 0.5);
      Yk1 := trunc(X[14] * Yfact + 0.5);
     for I := 2 to 14 do
    begin
       Xk2 := trunc(X[I] * Xfact + 0.5);
       Yk2 := trunc(X[14 - I + 1] * Yfact + 0.5);
      Xp1 := Xr - Xk1;
      Yp1 := Yr + Yk1;
      Xp2 := Xr - Xk2;
      Yp2 := Yr + Yk2;
       DrawLine(Xp1, Yp1, Xp2, Yp2);
      Xp1 := Xr + Xk1;
      Xp2 := Xr + Xk2;
       DrawLine(Xp1, Yp1, Xp2, Yp2);
      Yp1 := Yr - Yk1;
      Yp2 := Yr - Yk2;
       DrawLine(Xp1, Yp1 + 1, Xp2, Yp2 + 1);
      Xp1 := Xr - Xk1;
      Xp2 := Xr - Xk2;
       DrawLine(Xp1, Yp1 + 1, Xp2, Yp2 + 1);
      Xk1 := Xk2;
      Yk1 := Yk2;
    end;
  end
 else
   DP(Xr, Yr);
end; { DrawCircleDirect }

procedure DrawCircle{(X_R, Y_R, Xradius : real)};
begin
  DrawCircleDirect(WindowX(X_R), WindowY(Y_R), trunc(Xradius));
end; { DrawCircle }
```

```
procedure DrawCross{(X1, Y1, Scale : integer)};
begin
  DrawLineClipped(X1 - Scale, Y1, X1 + Scale + 2, Y1);
  DrawLineClipped(X1, Y1 - Scale, X1, Y1 + Scale + 1);
end; { DrawCross }

procedure DrawStar{(X, Y, Scale : integer)};
begin
  DrawLineClipped(X - Scale, Y + Scale, X + Scale + 1, Y - Scale - 1);
  DrawLineClipped(X - Scale, Y - Scale, X + Scale + 1, Y + Scale + 1);
  DrawLineClipped(X - Scale - 2, Y, X + Scale + 4, Y);
end; { DrawStar }

procedure DrawSquareC{(X1, Y1, X2, Y2 : integer; Fill : boolean)};
var
  I : integer;

procedure DSC(X1, X2, Y : integer);
begin
  DrawStraight(X1, X2, Y);
end; { DSC }

begin { DrawSquareC }
  if not Fill then
   begin
     DrawLineClipped(X1, Y1, X2, Y1);
     DrawLineClipped(X2, Y1, X2, Y2);
     DrawLineClipped(X1, Y2, X2, Y2);
     DrawLineClipped(X1, Y2, X1, Y1);
   end
  else
   for I := Y2 to Y1 do
     DSC(X1, X2, I);
end; { DrawSquareC }

procedure DrawSquare{(X1, Y1, X2, Y2 : real; Fill : boolean)};
var
  I, X1Loc, Y1Loc, X2Loc, Y2Loc : integer;
  DirectModeLoc : boolean;

procedure DS(X1, X2, Y : integer);
begin
  if LineStyleGlb = 0 then
    DrawStraight(X1, X2, Y)
  else
    DrawLine(X1, Y, X2, Y);
```

```
end; { DS }

procedure DSC(X1, X2, Y : integer);
begin
  DS(X1, X2, Y);
end; { DSC }

procedure DrawSqr(X1, Y1, X2, Y2 : integer; Fill : boolean);
var
  I : integer;
begin
  if not Fill then
   begin
     DS(X1, X2, Y1);
      DrawLine(X2, Y1, X2, Y2);
     DS(X1, X2, Y2);
      DrawLine(X1, Y2, X1, Y1);
   end
  else
   for I := Y1 to Y2 do
     DS(X1, X2, I);
end; { DrawSqr }

begin { DrawSquare }
  X1Loc := WindowX(X1);
  Y1Loc := WindowY(Y1);
  X2Loc := WindowX(X2);
  Y2Loc := WindowY(Y2);
  if not Fill then
   begin
     DSC(X1Loc, X2Loc, Y1Loc);
      DrawLineClipped(X2Loc, Y1Loc, X2Loc, Y2Loc);
     DSC(X1Loc, X2Loc, Y2Loc);
      DrawLineClipped(X1Loc, Y2Loc, X1Loc, Y1Loc);
   end
  else
   for I := Y1Loc to Y2Loc do
     DSC(X1Loc, X2Loc, I);
end; { DrawSquare }

procedure DrawAscii{(X, Y : integer; Size, CharByte : byte)};
begin
  MoveTo(X, Y);
  TextSize(Size * 12);
   DrawChar(Chr(CharByte));
end; { DrawAscii }
```

```
procedure DrawText{(X, Y, Scale : integer; Txt : WrkString)};
var
  Index : integer;
  EscStr : boolean;
  StringLen : integer;
  AsciiValue : integer;
  SymbolScale : integer;
  SymbolCode : integer;
begin
  Index := 1;
  EscStr := FALSE;
  StringLen := Length(Txt);
  while (Index <= StringLen) and (not EscStr) do
  begin
     if Txt[Index] = #27 then
     EscStr := TRUE;
    Index := Index + 1;
  end;
  if not EscStr then
    begin
      MoveTo(X, Y);
       TextSize(Scale * 12);
        DrawString(Txt);
    end
  else
    begin
      Index := 1;
      while Index <= StringLen do
     begin
         AsciiValue := Ord(Txt[Index]);
        if AsciiValue = 27 then
        begin
          SymbolScale := Scale;
           Index := Index + 1;
           if Index <= StringLen then
          begin
              SymbolCode := Ord(Txt[Index]) - 48;
                if (Index + 2 <= StringLen) and (Ord(Txt[Index + 1]) = 64) then
            begin
                SymbolCode := Ord(Txt[Index]) - 48;
              Index := Index + 2;
            end;
           case SymbolCode of
              1   : DrawCross(X + SymbolScale, Y + Scale, SymbolScale);
              2   : DrawCrossDiag(X + SymbolScale, Y + Scale, SymbolScale);
                3,4 : DrawSquareC(X, Y + (SymbolScale shl 1) - 1,
                             X + (SymbolScale shl 1), Y - 1, (SymbolCode = 4));
```

```
          5   : begin
                  DrawDiamond(X + trunc(1.5 * SymbolScale),
                          Y + SymbolScale - 1, SymbolScale + 1);
               X := X + SymbolScale;
              end;
            6   : DrawWye(X + SymbolScale, Y + SymbolScale - 1, SymbolScale);
          7   : begin
                  DrawStar(X + SymbolScale shl 1, Y + SymbolScale - 1, SymbolScale);
                X := X + SymbolScale shl 1;
              end;
             8   : DrawCircleDirect(X + SymbolScale, Y + (SymbolScale shr 1),
                           SymbolScale + 1);
        end;
         X := X + 3 * SymbolScale;
        SymbolScale := Scale;
       end;
      end
     else
       DrawAscii(X, Y, Scale, AsciiValue);
      Index := Index + 1;
    end;
   end;
end; { DrawText }

procedure DrawTextW{(X, Y : real; Scale : integer; Txt : WrkString)};
begin
  DrawText(WindowX(X), WindowY(Y), Scale, Txt);
end; { DrawTextW }

procedure TextStyle{(Face : Style)};
{ Face = (bold, italic, underline, outline, shadow, condense, extend) }
begin
 TextFace(Face);
end; { TextStyle }

procedure HardCopy{(TopWin : boolean)};
begin
  PrDrvrOpen;
 if TopWin then
   { Print the top folder. }
    PrCtlCall(iPrEvtCtl, LPrEvtTop, 0, LScreenBits)
 else
   { Print the whole screen. }
    PrCtlCall(iPrEvtCtl, LPrEvtAll, 0, LScreenBits);
  PrDrvrClose;
end; { HardCopy }
```

```
procedure OpenPic{(WinNum : integer; ShowPic : boolean)};
begin
  if Window[WinNum].H <> NIL then
  begin
      KillPicture(Window[WinNum].H);
    Window[WinNum].H := NIL;
  end;
   RectRgn(Window[WinNum].W.Port.clipRgn, ScreenBits.bounds);
   Window[WinNum].H := OpenPicture(Window[WinNum].W.Port.PortRect);
  if ShowPic then
   ShowPen
end; { OpenPic }

procedure DrawPic{(WinNum : integer)};
var
  PictRect : Rect;
begin
   PictRect := Window[WinNum].W.Port.PortRect;
  with PictRect do
  begin
     if ((Bottom - Top) < 200) OR ((Right - Left) < 200) then
   begin
      Right := Right - 16;  { so we don't overwrite the grow region }
     Bottom := Bottom - 16;{ on a small window.                     }
   end;
  end;
   DrawPicture(Window[WinNum].H, PictRect);
end; { DrawPic }

procedure ErasePic{(WinNum : integer)};
begin
  if Window[WinNum].H <> NIL then
  begin
      KillPicture(Window[WinNum].H);
    Window[WinNum].H := NIL;
  end;
end; { ErasePic }

procedure ClearWindow{(WinNum : integer)};
begin
   EraseRect(Window[WinNum].W.Port.PortRect);
end; { ClearWindow }

function WhereX{ : integer};
var
  Pt : Point;
begin
```

```
  GetPen(Pt);
  WhereX := Pt.H;
end; { WhereX }

function WhereY{ : integer};
var
  Pt : Point;
begin
  GetPen(Pt);
  WhereY := Pt.V;
end; { WhereY }

procedure SetWindow{(X1, Y1, X2, Y2 : integer)};
begin
  X1RefGlb := X1;
  Y1RefGlb := Y1;
  X2RefGlb := X2;
  Y2RefGlb := Y2;
  BxGlb := (X2 - X1) / (X2WldGlb - X1WldGlb);
  ByGlb := (Y2 - Y1) / (Y2WldGlb - Y1WldGlb);
  AxGlb := X1 - X1WldGlb * BxGlb;
  AyGlb := Y1 - Y1WldGlb * ByGlb;
  AxisGlb := FALSE;
end; { SetWindow }

procedure FindWorld{(I : integer; A : PlotArray; NPoints : integer)};

var
  J : integer;
  Xmax, Ymax, Xmin, Ymin, Xmid, Ymid, Xdiff, Ydiff : real;

begin
  NPoints := abs(NPoints);
  if NPoints > 2 then
    if I in [1..MaxWorldsGlb] then
    begin
      Xmax := A[1, 1];
      Ymax := A[1, 2];
      Xmin := Xmax;
      Ymin := Ymax;
      for J := 2 to NPoints do
      begin
        if A[J, 1] > Xmax then
          Xmax := A[J, 1]
        else
          if A[J, 1] < Xmin then
            Xmin := A[J, 1];
```

```
          if A[J, 2] > Ymax then
            Ymax := A[J, 2]
         else
            if A[J, 2] < Ymin then
              Ymin := A[J, 2];
       end;
         Xmin := Round(Xmin);
          Ymin := Round(Ymin) - 0.5;
        Xmax := Round(Xmax);
         Ymax := Round(Ymax) + 0.5;
          DefineWorld(I, Xmin, Ymin, Xmax, Ymax);
          SelectWorld(I);
     end
   else
          Error('FindWorld #1')
  else
       Error('FindWorld # 2');
end; { FindWorld }

procedure FindWorld1{(I : integer; A : PlotArray; NPoints : integer)};

var
  J : integer;
  Xmax, Ymax, Xmin, Ymin, Xmid, Ymid, Xdiff, Ydiff : real;

begin
        Xmin := XMn;
         Ymin := YMn{Round(YMn)} ;
         Xmax := XMx{Round(XMx)};
         Ymax := YMx{Round(YMx)} ;
          DefineWorld(I, Xmin, Ymin, Xmax, Ymax);
          SelectWorld(I);
 end; { FindWorld }

procedure DrawAxis{(Footer1, Footer2 : WrkString; Arrows : boolean)};
var
  LineStyleLoc, Xk0, Yk0, Xk1, Yk1, Xk2, Yk2,
  MaxExponentX, MaxExponentY, TickPoint : integer;
  TickSmall, TickLarge, Max, Min, Tick, Offset, Diff : real;

function Log(X : real) : real;
{ Base 10 logarithm of X. }
begin
  Log := Ln(X) / Ln(10.0);
end; { Log }

function ALog(X : real) : real;
```

```
{ Ten raised to the X power. }
begin
  ALog := Exp(X * Ln(10.0));
end; { Alog }

function Frac(R : real) : real;
{ Return the fractional part of the real number R. }
begin
  Frac := R - Int(R);
end; { Frac }

procedure Ticks(NTicks : integer; Max, Min : real;
          var TickSmall, TickLarge : real);
{ NTicks    : The approximate number of tick marks in the interval. }
{ Max, Min : World coordinates of the axis extremes.              }
{ TickSmall, TickLarge :Tick mark intervals, in world coordinates .}
var
  TickLog : array[1..4] of real;
  I, J, ChA : integer;
  Delta, XTicks, LogTicks, Mant, MinDiff, Diff : real;
begin
  TickLog[1] := 0.0;
  TickLog[2] := Log(2.0);
  TickLog[3] := Log(5.0);
  TickLog[4] := 1.0;
  XTicks := NTicks;
  Delta := Max - Min;
  XTicks := Delta / XTicks;
  LogTicks := Log(XTicks);
  ChA := trunc(LogTicks);
  if LogTicks < 0.0 then
   ChA := ChA - 1;
  MinDiff := 1.0;
  Mant := LogTicks - ChA; { Fractional part of logarithm }
  for I := 1 to 4 do
  begin
    Diff := Abs(Mant - TickLog[I]);
    if (Diff < MinDiff) then
   begin
     MinDiff := Diff;
     J := I;
   end;
  end;
  LogTicks := ChA + TickLog[J]; { Logarithm of tick mark }
  TickLarge := ALOg(LogTicks); { The tick mark }

  { Find the small tick marks, that are two tick scales smaller }
```

```
  if J > 2 then
   begin
     J := J - 2;
     LogTicks := ChA + TickLog[J];
   end
  else
   begin
     J := J + 1;
     LogTicks := ChA + TickLog[J] - 1;
   end;
  TickSmall := ALog(LogTicks);
end; { Ticks }

function StringNumber(X1 : real; MaxExponent : integer) : WrkString;
begin
  StringNumber := RealToStr(X1 * Exp(-MaxExponent * Ln(10.0)));
end; { StringNumber }

function GetExponent(X1 : real) : integer;
begin
  GetExponent := 0;
  if X1 <> 0.0 then
    if ABS(X1) >= 1.0 then
       GetExponent := trunc(Ln(ABS(X1)) / Ln(10.0))
   else
       GetExponent := -trunc(ABS(Ln(ABS(X1))) / Ln(10.0) + 1.0);
end; { GetExponent }

procedure DrawNum(X1, Y1, MaxExponent : integer; Number : real);
var
  StrNumber : WrkString;
begin
  TextSize(9);
  StrNumber := StringNumber(Number, MaxExponent);
  Y1 := Y1 - 3;
  MoveTo(X1, Y1);
  DrawString(StrNumber);
  TextSize(12);
end; { DrawNum }

procedure DrawExponent(X1, Y1, MaxExponent : integer);
var
  NumStr : WrkString;
begin
  MoveTo(X1, Y1);
  TextSize(9);
  DrawChar('x');
```

```
   DrawChar(' ');
   DrawChar('1');
   DrawChar('0');
  X1 := WhereX + 1;
  Y1 := WhereY - 3;
  MoveTo(X1, Y1);
  NumStr := IntToStr(MaxExponent);
  TextSize(7);
   DrawString(NumStr);
  TextSize(12);
end; { DrawExponent }

begin { DrawAxis }
  LineStyleLoc := LinestyleGlb;
  SetLineStyle(0);  { Black }

  Xk0 := X1RefGlb + X1Offset;
  Yk0 := Y2RefGlb - Y2Offset;
  Xk1 := Xk0;
  Yk1 := Y1RefGlb + Y1Offset;
  Xk2 := X2RefGlb - X2Offset;
  Yk2 := Yk0;

  MoveTo(XK0, YK0);      { Draw the Y axis with optional Arrows }
   LineTo(XK1, YK1);
  if Arrows then
  begin
    MoveTo(Xk0, Yk1);
     LineTo(Xk0 - 4, Yk1 + 4);
    MoveTo(Xk0, Yk1);
     LineTo(Xk0 + 4, Yk1 + 4);
     DP(Xk0, Yk1 - 1);
  end;

  MoveTo(Xk0, Yk0);      { Draw the X axis with optional Arrows }
   LineTo(Xk2 + 1, Yk2);
  if Arrows then
  begin
    MoveTo(Xk2, Yk2);
     LineTo(Xk2 - 4, Yk2 - 4);
    MoveTo(Xk2, Yk2);
     LineTo(Xk2 - 4, Yk2 + 4);
  end;

  if Footer1 <> '' then  { Draw the 1st footer below the X axis }
  begin
    MoveTo(Xk0, Yk0 + 45);
```

```
    TextSize(9);
     DrawString(Footer1);
end;

 if Footer2 <> '' then  { Draw the 2nd footer below the X axis }
begin
    MoveTo(Xk0, Yk0 + 65);
    TextSize(9);
     DrawString(Footer2);
end;

 if (ABS(Yk0 - Yk1) >= 35) and (ABS(Xk2 - Xk1) >= 150) then
begin
   if ABS(Y2WldGlb) > ABS(Y1WldGlb) then
     MaxExponentY := GetExponent(Y2WldGlb)
  else
     MaxExponentY := GetExponent(Y1WldGlb);

  if MaxExponentY <> 0 then { Draw the power of ten on top of Y axis }
      DrawExponent(Xk1 - 30, Yk1 + 2, MaxExponentY);

   TickPoint := Yk0;
  if Y1WldGlb > Y2WldGlb then
    begin
       Max := Y1WldGlb;
       Min := Y2WldGlb;
    end
  else
    begin
       Max := Y2WldGlb;
       Min := Y1WldGlb;
    end;

   { Using the Max and Min values, this procedure call calculates }
  { large and small Tick Marks in world coordinates.              }
    Ticks(5, Max, Min, TickSmall, TickLarge);

   Offset := Min / TickLarge;
   Offset := Offset - Frac(Offset);
   Tick := Offset * TickLarge;
   if Tick < Min then
     Tick := Tick + TickLarge;

   { Tick is the world coordinate at which the tick mark is to be drawn }
    Diff := Max - Min;
   { Plot large tick marks and Numeric labels }
  while Tick <= Max do
```

```
begin
    TickPoint := Yk0 - Trunc((Yk0 - Yk1) * (Tick - Min) / Diff);
   MoveTo(Xk0, TickPoint);
   LineTo(Xk0 - 4, TickPoint);
   DrawNum(X1Offset - 30, TickPoint + 7, MaxExponentY, Tick);
  Tick := Tick + TickLarge;
end;

 { The same repeated for the small tick marks, }
 { only without axis numbering.               }
 Offset := Min / TickSmall;
 Offset := Offset - Frac(Offset);
 Tick := Offset * TickSmall;
 if Tick < Min then
   Tick := Tick + TickSmall;
 while (Tick + 0.01) < Max do
begin
    TickPoint := Yk0 - Trunc((Yk0 - Yk1) * (Tick - Min) / Diff);
   MoveTo(Xk0, TickPoint);
   LineTo(Xk0 - 2, TickPoint);
   Tick := Tick + TickSmall;
end;

 if ABS(X2WldGlb) > ABS(X1WldGlb) then
  MaxExponentX := GetExponent(X2WldGlb)
else
  MaxExponentX := GetExponent(X1WldGlb);
 if MaxExponentX <> 0 then { Draw power of ten label on X axis }
   DrawExponent(Xk2 - 25, Yk0 + 28, MaxExponentX);

 { This is the same as for the Y axis, but the window }
 { and world are appropriate for the X axis.          }
 TickPoint := Xk0;
 if X1WldGlb > X2WldGlb then
  begin
   Max := X1WldGlb;
   Min := X2WldGlb;
  end
else
  begin
   Max := X2WldGlb;
   Min := X1WldGlb;
  end;
  Ticks(5, Max, Min, TickSmall, TickLarge);
 Offset := Min / TickLarge;
 Offset := Offset - Frac(Offset);
 Tick := Offset * TickLarge;
```

```
    if Tick < Min then
      Tick := Tick + TickLarge;
     Diff := Max - Min;
    while Tick <= Max do
    begin
        TickPoint := Xk0 + Trunc((Xk2 - Xk0) * (Tick - Min) / Diff);
       MoveTo(TickPoint, Yk0);
       LineTo(TickPoint, Yk0 + 4);
        DrawNum(TickPoint - 14, Yk0 + 20, MaxExponentX, Tick);
      Tick := Tick + TickLarge;
    end;
     Offset := Min / TickSmall;
     Offset := Offset - Frac(Offset);
     Tick := Offset * TickSmall;
    if Tick < Min then
      Tick := Tick + TickSmall;
     while (Tick + 0.01) < Max do
    begin
        TickPoint := Xk0 + Trunc((Xk2 - Xk0) * (Tick - Min) / Diff);
       MoveTo(TickPoint, Yk0);
       LineTo(TickPoint, Yk0 + 2);
      Tick := Tick + TickSmall;
    end;
  end;
   SetLineStyle(LineStyleLoc);
  AxisGlb := TRUE;
end; { DrawAxis }

procedure ResetAxis;
begin
  AxisGlb := true;
end; { ResetAxis }

procedure DrawPolygon{(A : PlotArray; First, NPoints, Line, Scale,
                    Lines : integer; CrossHairs : boolean)};
var
  i, X1, X2, Y1, Y2, XOffset, YOffset,
  X1RefLoc, Y1RefLoc, X2RefLoc, Y2RefLoc,
  DeltaY, XOs1, XOs2, YOs1, YOs2 : integer;
  AutoClip, DirectModeLoc, PlotLine, PlotSymbol, Flipped : boolean;
  X1Loc, Y1Loc, X2Loc, Y2Loc : integer;
  Temp : real;
  LineStyleLoc2 : integer;
  DrawPt : Boolean;

procedure DrawPointClipped(X, Y : integer);
begin
```

```
  if (X1 > X1RefGlb) and (X2 < X2RefGlb) then
    if (Y1 > Y1RefGlb) and (Y2 < Y2RefGlb) then
      DP(X, Y);
end; { DrawPointClipped }

procedure DrawItem(X, Y : integer);
var
  LineStyleLoc : integer;
begin
  LineStyleLoc := LineStyleGlb;
  SetLineStyle(0); { Black }
  case Line of
    2    : DrawCrossDiag(X, Y, Scale);
    3, 4 : DrawSquareC(X - Scale, Y + Scale, X + Scale, Y - Scale, (Line = 4));
    5    : DrawDiamond(X, Y, Scale + 1);
    6    : DrawWye(X, Y, Scale + 1);
    1    : DrawCross(X, Y, Scale);
    8    : DrawCircleDirect(X, Y, Scale + 1);
    9    : begin
             PlotLine := false;
             if AutoClip then
               DrawPointClipped(X, Y)
             else
               DP(X, Y);
           end;
    7    : DrawPoint(X, Y){, Scale)};
  end;
  SetLineStyle(LineStyleLoc);
end; { DrawItem }

begin { DrawPolygon }
  if AxisGlb then
    Flipped := FALSE
  else
    begin
      Flipped := TRUE;
      Temp := World[WorldNdxGlb].Y1;
      World[WorldNdxGlb].Y1 := World[WorldNdxGlb].Y2;
      World[WorldNdxGlb].Y2 := Temp;
      SelectWorld(WorldNdxGlb);
      SelectWind(WindowNdxGlb, TRUE);
    end;
  if abs(NPoints - First) >= 2 then
    begin
      AutoClip := (NPoints < 0);
      NPoints := abs(NPoints);
      XOs1 := 1;
```

```
XOs2 := 1;
YOs1 := 6;
YOs2 := 6;
 if AxisGlb then
begin
  XOs1 := X1Offset;
  XOs2 := X2Offset;
  YOs1 := Y1Offset;
  YOs2 := Y2Offset;
   if  ((X2RefGlb - XOs2 - X1RefGlb + XOs1) > (XOs1 + XOs2)) and
       ((Y2RefGlb - YOs2 - Y1RefGlb + YOs1) > (YOs1 + YOs2)) then
  begin
    X1RefLoc := X1RefGlb;
    X1 := X1RefGlb + XOs1;
    Y1RefLoc := Y1RefGlb;
    Y1 := Y1RefGlb + YOs1;
    X2RefLoc := X2RefGlb;
    X2 := X2RefGlb - XOs2;
    Y2RefLoc := Y2RefGlb;
    Y2 := Y2RefGlb - YOs2;
     SetWindow(X1, Y1, X2, Y2);
    AxisGlb := TRUE;
  end;
end;
  PlotLine := (Line >= 0);
  PlotSymbol := (Line <> 0);
  Line := abs(Line);
 Scale := abs(Scale);
 if Lines < 0 then
     DeltaY := Trunc(1.0 / (abs(Y1WldGlb) + abs(Y2WldGlb)) *
                        abs(Y1WldGlb) * abs(Y2RefGlb - Y1RefGlb)) + 1
else
  DeltaY := 0;
 if (NPoints < 2) then
     Error('DrawPolygon #1')
else
  begin
    if CrossHairs then
   begin
      LineStyleLoc2 := LineStyleGlb;
       SetLineStyle(3); { Light Gray }
       MoveTo(X1RefGlb, Y1RefGlb + Y2RefGlb - WindowY(0.0));
       LineTo(X2RefGlb, Y1RefGlb + Y2RefGlb - WindowY(0.0));
       MoveTo(WindowX(0.0), Y1RefGlb);
       LineTo(WindowX(0.0), Y2RefGlb);
       SetLineStyle(LineStyleLoc2);
   end;
```

```
X1 := WindowX(A[First, 1]);
Y1 := Y1RefGlb + Y2RefGlb - WindowY(A[First, 2]) ;
Drawltem(X1, Y1);
if Abs(Lines) = 1 then
 if AutoClip then
   DrawLineClipped(X1, Y2RefGlb - DeltaY, X1, Y1)
else
 begin
   MoveTo(X1, Y2RefGlb - DeltaY);
   LineTo(X1, Y1);
 end;
DrawPt := True;
for I:= First + 1 to NPoints do
begin
   X2 := WindowX(A[I, 1]);
   Y2 := Y2RefGlb + Y1RefGlb - WindowY(A[I, 2]);
   Drawltem(X2, Y2);
 if StepA then begin
  if not AMarkStatus then
  begin
     if (A[I, 2] > 0) and DrawPt   then {Clip(X1,Y1,X2,Y2)}
   begin
     if ARJustification then
    begin
       MoveTo(X2+15, Y2+5);
      TextSize(9);
       DrawString(Ds);
    end
    else
    begin
       MoveTo(X2-65, Y2);
      TextSize(9);
       DrawString(Ds);
    end;
    DrawPt := False;
   end;
 end
 else
  begin
     if (I > (NPoints - InitDegree)) and (A[I, 2] > 0) and DrawPt   then
   begin
     if ARJustification then
    begin
       MoveTo(X2+15, Y2+5);
      TextSize(9);
       DrawString(Ds);
    end
```

```
    else
     begin
         MoveTo(X2-65, Y2);
        TextSize(9);
         DrawString(Ds);
     end;
     DrawPt := False;
   end;
 end;
 end;{StepA}
if not StepA then begin
 if not BMarkStatus then
 begin
      if (A[I, 2] > 0) and DrawPt    then {Clip(X1,Y1,X2,Y2)}
   begin
      if BRJustification then
     begin
         MoveTo(X2+15, Y2+5);
        TextSize(9);
         DrawString(Ds);
     end
    else
     begin
         MoveTo(X2-65, Y2);
        TextSize(9);
         DrawString(Ds);
     end;
     DrawPt := False;
   end;
 end
else
 begin
      if (I > (NPoints - InitDegree)) and (A[I, 2] > 0) and DrawPt    then
   begin
      if BRJustification then
     begin
         MoveTo(X2+15, Y2+5);
        TextSize(9);
         DrawString(Ds);
     end
    else
     begin
         MoveTo(X2-65, Y2);
        TextSize(9);
         DrawString(Ds);
     end;
     DrawPt := False;
```

```
           end;
          end;
          end;{not StepA}

           if Abs(Lines) = 1 then
            if AutoClip then
               DrawLineClipped(X2, Y2RefGlb - DeltaY, X2, Y2)
          else
            begin
               MoveTo(X2, Y2RefGlb - DeltaY);
               LineTo(X2, Y2);
            end;
          if PlotLine then
            if AutoClip then
               DrawLineClipped(X1, Y1, X2, Y2)
          else
            begin
               MoveTo(X1, Y1);
               LineTo(X2, Y2);
            end;
         X1 := X2;
         Y1 := Y2;
       end;
      end;
     if AxisGlb then
    begin
      SetWindow(X1RefLoc, Y1RefLoc, X2RefLoc, Y2RefLoc);
      AxisGlb := false;
    end;
  end
 else
     Error('DrawPolygon # 1');
 if Flipped then
 begin
    Temp := World[WorldNdxGlb].Y1;
     World[WorldNdxGlb].Y1 := World[WorldNdxGlb].Y2;
    World[WorldNdxGlb].Y2 := Temp;
     SelectWorld(WorldNdxGlb);
    SelectWind(WindowNdxGlb, TRUE);
 end;
end; { DrawPolygon }

procedure Hatch{(X_1, Y_1, X_2, Y_2, Delta : real)};
var
 X1, Y1, X2, Y2 : integer;

procedure HatchDirect(X1, Y1, X2, Y2, Delta : integer);
```

```
var
  I, Yst, Yen, Count : integer;
  X1RefLoc, X2RefLoc, Y1RefLoc, Y2RefLoc : integer;
  ClippingLoc : boolean;
  X1D, Y1D, X2D, Y2D : integer;

begin { HatchDirect }
  if Delta <> 0 then
  begin
    HatchGlb := true;
    ClippingLoc := ClippingGlb;
    ClippingGlb := true;
    X1RefLoc := X1RefGlb;
    X1RefGlb := X1;
    X2RefLoc := X2RefGlb;
    X2RefGlb := X2;
    Y1RefLoc := Y1RefGlb;
    Y1RefGlb := Y1;
    Y2RefLoc := Y2RefGlb;
    Y2RefGlb := Y2;
    Yst := Y1 + Delta;
    Yen := Y1 - X2 + X1 + Delta;
    if Delta < 0 then
    begin
      Delta := -Delta;
      I := Yst;
      Yst := Yen;
      Yen := I;
    end;
    Count := (Y2 - Y1 + X2 - X1 + X2 - X1) div Delta;
    for I := 1 to Count-1 do
    begin
      X1D := X1;
      Y1D := Yst;
      X2D := X2;
      Y2D := Yen;
      if Clip(X1D, Y1D, X2D, Y2D) then
      begin
        MoveTo(X1D, Y1D);
        LineTo(X2D, Y2D);
      end;
      Yst := Yst + Delta;
      Yen := Yen + Delta;
    end;
    ClippingGlb := ClippingLoc;
    HatchGlb := false;
    X1RefGlb := X1RefLoc;
```

```
    X2RefGlb := X2RefLoc;
    Y1RefGlb := Y1RefLoc;
    Y2RefGlb := Y2RefLoc;
  end;
end; { HatchDirect }

begin { Hatch }
      HatchDirect(trunc(X_1), trunc(Y_1), trunc(X_2), trunc(Y_2), trunc(Delta))
end; { Hatch }

procedure DrawHistogram{(A :PlotArray; NPoints : integer;
                    Hatching : boolean; HatchStyle : integer)};

var
  X1, X2, Y2, NPixels, Delta, NDiff, YRef, LineStyleLoc, I : integer;
  Fract, S, Y : real;
  DirectModeLoc, Negative : boolean;
  X1Loc, Y1Loc, X2Loc, Y2Loc : integer;
  X1RefLoc, Y1RefLoc, X2RefLoc, Y2RefLoc, YAxis : integer;
  Temp : real;

begin { DrawHistogram }
  if ABS(NPoints) >= 2 then
   begin
     LineStyleLoc := LinestyleGlb;
      SetLineStyle(0); { Black }
     if AxisGlb then
    begin
       X1RefLoc := Window[WindowNdxGlb].X1;
       Y1RefLoc := Window[WindowNdxGlb].Y1;
       X2RefLoc := Window[WindowNdxGlb].X2;
       Y2RefLoc := Window[WindowNdxGlb].Y2;
       SetWindow(X1RefGlb + X1Offset, Y1RefGlb + Y1Offset,
              X2RefGlb - X2Offset, Y2RefGlb - Y2Offset);
     AxisGlb := TRUE;
    end;
     Negative := NPoints < 0;
      NPoints := ABS(NPoints);
     NPixels := X2RefGlb - X1RefGlb;
     Delta := NPixels div NPoints;
      NDiff := NPixels - Delta * NPoints;
      Fract := NDiff / NPoints;
      S := -Fract;
     X1 := X1RefGlb;
     Temp := Y2RefGlb + Y1RefGlb - AyGlb;
     if Temp > MaxInt then
       Temp := MaxInt
```

```
else
   if Temp < -32767 then
     Temp := -32767;
 YRef := trunc(Temp);
if Negative then
   DrawStraight(X1, X2RefGlb, YRef);
YAxis := Y1RefGlb;
if BYGlb > 0 then
  YAxis := Y2RefGlb;
for I := 1 to NPoints do
begin
  X2 := X1 + Delta;
  Y := A[I, 2];
  if not Negative then
   Y := ABS(Y);
  Temp := AyGlb + ByGlb * Y;
  if Temp > MaxInt then
   Temp := MaxInt
 else
    if Temp < -32767 then
      Temp := -32767;
   Y2 := Y2RefGlb + Y1RefGlb - trunc(Temp);
  if not Negative then
   begin
      MoveTo(X1, YAxis);
       LineTo(X1, Y2);
      MoveTo(X1, Y2);
       LineTo(X2, Y2);
      MoveTo(X2, Y2);
       LineTo(X2, YAxis);
      if Hatching then
        if Odd(I) then
           Hatch(X1, Y2, X2, YAxis, HatchStyle)
       else
            Hatch(X1, Y2, X2, YAxis, -HatchStyle);
  end
 else
   begin
      MoveTo(X1, YRef);
       LineTo(X1, Y2);
      MoveTo(X1, Y2);
       LineTo(X2, Y2);
      MoveTo(X2, Y2);
      LineTo(X2, YRef);
      if Hatching then
        if YRef - Y2 < 0 then
          if Odd(I) then
```

```
              Hatch(X1, YRef, X2, Y2, HatchStyle)
            else
               Hatch(X1, YRef, X2, Y2, -HatchStyle)
          else
            if Odd(I) then
              Hatch(X1, Y2, X2,YRef, HatchStyle)
            else
               Hatch(X1, Y2, X2, YRef, -HatchStyle);
        end;
       X1 := X2;
     end;
      if AxisGlb then
     begin
       SetWindow(X1RefLoc, Y1RefLoc, X2RefLoc, Y2RefLoc);
      AxisGlb := FALSE;
     end;
       SetLineStyle(LineStyleLoc);
   end
  else
      Error('DrawHistogram');
end; { DrawHistogram }

begin
end.  { TurboGraph }
```

```
* This is the resource file that defines the menus and icons for MacRootLocus.

MacRootLocus.Rsrc

TYPE DLOG

  ,256 (36)
CE Parameter Dialog
70 100 300 412
Visible NoGoAway
1
0
256

TYPE DITL
  ,256 (36)
11

BtnItem Enabled
185 240 210 300
Cancel

BtnItem Enabled
150 240 175 300
OK

EditTextItem Enabled
60 260 75 280

EditTextItem Enabled
95 165 110 205
1

EditTextItem Enabled
120 165 135 205
100

EditTextItem Enabled
145 165 160 205
1E-6

StatText Disabled
20 15 40 250
Characteristic Equation Parameter

StatText Disabled
60 40 75 240
```

Degree of the polynomial

StatText Disabled
95 40 110 150
InitGuess

StatText Disabled
120 40 135 150
Maxiter

StatText Disabled
145 40 160 150
Tolerance

TYPE DLOG

,257 (36)
One Parameter Root Locus Plot Data
50 86 315 426
Visible NoGoAway
1
0
257

TYPE DITL
,257 (36)
21
BtnItem Enabled
35 270 55 330
Cancel

BtnItem Enabled
10 270 30 330
PLOT

RadioItem Enabled
95 40 110 265
Linear Point Interval

RadioItem Enabled
115 40 130 265
Logarithmic Point Interval

RadioItem Enabled
145 25 160 150

Auto Scale Axis

RadioItem Enabled
145 165 160 320
Manual Scale Axis

EditTextItem Enabled
65 110 80 160
0.1

EditTextItem Enabled
65 265 80 315
10000

EditTextItem Enabled
170 85 185 125
-10

EditTextItem Enabled
170 225 185 265
5

EditTextItem Enabled
195 85 210 125
-10

EditTextItem Enabled
195 225 210 265
10

EditTextItem Enabled
230 135 245 165
50

StatText Disabled
20 10 40 255
One Parameter Root Locus Plot Data

StatText Disabled
65 25 80 100
AMin Gain

StatText Disabled
65 180 80 260
AMax Gain

StatText Disabled

170 25 185 80
X Min

StatText Disabled
170 165 185 220
XMax

StatText Disabled
195 25 210 80
Y Min

StatText Disabled
195 165 210 220
YMax

StatText Disabled
230 25 245 125
Points To Plot

TYPE DLOG

,258 (36)
Two Parameter Root Locus Plot Data
30 86 330 426
Visible NoGoAway
1
0
258

TYPE DITL
,258 (36)
35

BtnItem Enabled
35 270 55 330
Cancel

BtnItem Enabled
10 270 30 330
Plot

RadioItem Enabled
120 40 135 265
Linear Point Interval

RadioItem Enabled

140 40 155 265
Logarithmic Point Interval

RadioItem Enabled
250 130 265 180
Start

RadioItem Enabled
250 185 265 230
End

RadioItem Enabled
250 235 265 288
Right

RadioItem Enabled
250 293 265 338
Left

RadioItem Enabled
270 130 285 180
Start

RadioItem Enabled
270 185 285 230
End

RadioItem Enabled
270 235 285 288
Right

RadioItem Enabled
270 293 285 338
Left

EditTextItem Enabled
65 110 80 160
0.1

EditTextItem Enabled
65 265 80 315
10000

EditTextItem Enabled
90 110 105 160
0.1

EditTextItem Enabled
90 265 105 315
10000

EditTextItem Enabled
170 135 185 165
5

EditTextItem Enabled
170 285 185 315
50

EditTextItem Enabled
195 85 210 125
-10

EditTextItem Enabled
195 225 210 265
5

EditTextItem Enabled
220 85 235 125
-10

EditTextItem Enabled
220 225 235 265
10

StatText Disabled
20 10 40 255
Two Parameter Root Locus Plot Data

StatText Disabled
65 25 80 105
AMin Gain

StatText Disabled
65 180 80 260
AMax Gain

StatText Disabled
90 25 105 105
BMin Gain

StatText Disabled
90 180 105 260
BMax Gain

StatText Disabled
195 25 210 80
X Min

StatText Disabled
195 165 210 220
XMax

StatText Disabled
220 25 235 80
Y Min

StatText Disabled
220 165 235 220
YMax

StatText Disabled
170 25 185 125
How Many Loci

StatText Disabled
170 185 185 275
Points To Plot

StatText Disabled
250 25 265 125
AMark Point

StatText Disabled
270 25 285 125
BMark Point

TYPE DLOG

,301 (36)
Characteristic Equation Coefficient Data
105 56 260 456
Visible NoGoAway
1
0
301

TYPE DITL
,301
7

```
BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380
Cancel

EditTextItem Enabled
110 145 130 255

EditTextItem Enabled
110 270 130 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
85 148 105 208
S**1

StateTextItem Enabled
85 273 105 333
S**0


TYPE DLOG

   ,302 (36)
Characteristic Equation Coefficient Data
105 56 260 456
Visible NoGoAway
1
0
302

TYPE DITL
   ,302
9

BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380
```

Cancel

EditTextItem Enabled
110 20 130 130

EditTextItem Enabled
110 145 130 255

EditTextItem Enabled
110 270 130 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
85 23 105 83
S**2

StateTextItem Enabled
85 148 105 208
S**1

StateTextItem Enabled
85 273 105 333
S**0

TYPE DLOG

,303 (36)
Characteristic Equation Coefficient Data
75 56 285 456
Visible NoGoAway
1
0
303

TYPE DITL
,303
11

BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380

Cancel

EditTextItem Enabled
110 270 130 380

EditTextItem Enabled
165 20 185 130

EditTextItem Enabled
165 145 185 255

EditTextItem Enabled
165 270 185 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
85 273 105 333
S**3

StateTextItem Enabled
140 23 160 83
S**2

StateTextItem Enabled
140 148 160 208
S**1

StateTextItem Enabled
140 273 160 333
S**0

TYPE DLOG

 ,304 (36)
Characteristic Equation Coefficient Data
75 56 285 456
Visible NoGoAway
1
0
304

TYPE DITL
 ,304
13

```
BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380
Cancel

EditTextItem Enabled
110 145 130 255

EditTextItem Enabled
110 270 130 380

EditTextItem Enabled
165 20 185 130

EditTextItem Enabled
165 145 185 255

EditTextItem Enabled
165 270 185 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
85 148 105 208
S**4

StateTextItem Enabled
85 273 105 333
S**3

StateTextItem Enabled
140 23 160 83
S**2

StateTextItem Enabled
140 148 160 208
S**1

StateTextItem Enabled
140 273 160 333
S**0
```

TYPE DLOG

,305 (36)
Characteristic Equation Coefficient Data
75 56 285 456
Visible NoGoAway
1
0
305

TYPE DITL
,305
15

BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380
Cancel

EditTextItem Enabled
110 20 130 130

EditTextItem Enabled
110 145 130 255

EditTextItem Enabled
110 270 130 380

EditTextItem Enabled
165 20 185 130

EditTextItem Enabled
165 145 185 255

EditTextItem Enabled
165 270 185 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
85 23 105 83

S**5

StateTextItem Enabled
85 148 105 208
S**4

StateTextItem Enabled
85 273 105 333
S**3

StateTextItem Enabled
140 23 160 83
S**2

StateTextItem Enabled
140 148 160 208
S**1

StateTextItem Enabled
140 273 160 333
S**0

TYPE DLOG

,306 (36)
Characteristic Equation Coefficient Data
45 56 310 456
Visible NoGoAway
1
0
306

TYPE DITL
,306
17

BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380
Cancel

EditTextItem Enabled
110 270 130 380

EditTextItem Enabled
165 20 185 130

EditTextItem Enabled
165 145 185 255

EditTextItem Enabled
165 270 185 380

EditTextItem Enabled
220 20 240 130

EditTextItem Enabled
220 145 240 255

EditTextItem Enabled
220 270 240 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
85 273 105 333
S**6

StateTextItem Enabled
140 23 160 83
S**5

StateTextItem Enabled
140 148 160 208
S**4

StateTextItem Enabled
140 273 160 333
S**3

StateTextItem Enabled
195 23 215 83
S**2

StateTextItem Enabled
195 148 215 208
S**1

StateTextItem Enabled

195 273 215 333
S**0

TYPE DLOG

,307 (36)
Characteristic Equation Coefficient Data
45 56 310 456
Visible NoGoAway
1
0
307

TYPE DITL
,307
19

BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380
Cancel

EditTextItem Enabled
110 145 130 255

EditTextItem Enabled
110 270 130 380

EditTextItem Enabled
165 20 185 130

EditTextItem Enabled
165 145 185 255

EditTextItem Enabled
165 270 185 380

EditTextItem Enabled
220 20 240 130

EditTextItem Enabled
220 145 240 255

EditTextItem Enabled

220 270 240 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
85 148 105 208
S**7

StateTextItem Enabled
85 273 105 333
S**6

StateTextItem Enabled
140 23 160 83
S**5

StateTextItem Enabled
140 148 160 208
S**4

StateTextItem Enabled
140 273 160 333
S**3

StateTextItem Enabled
195 23 215 83
S**2

StateTextItem Enabled
195 148 215 208
S**1

StateTextItem Enabled
195 273 215 333
S**0

TYPE DLOG

,308 (36)
Characteristic Equation Coefficient Data
45 56 310 456
Visible NoGoAway
1
0
308

```
TYPE DITL
  ,308
21

BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380
Cancel

EditTextItem Enabled
110 20 130 130

EditTextItem Enabled
110 145 130 255

EditTextItem Enabled
110 270 130 380

EditTextItem Enabled
165 20 185 130

EditTextItem Enabled
165 145 185 255

EditTextItem Enabled
165 270 185 380

EditTextItem Enabled
220 20 240 130

EditTextItem Enabled
220 145 240 255

EditTextItem Enabled
220 270 240 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
85 23 105 83
S**8
```

StateTextItem Enabled
85 148 105 208
S**7

StateTextItem Enabled
85 273 105 333
S**6

StateTextItem Enabled
140 23 160 83
S**5

StateTextItem Enabled
140 148 160 208
S**4

StateTextItem Enabled
140 273 160 333
S**3

StateTextItem Enabled
195 23 215 83
S**2

StateTextItem Enabled
195 148 215 208
S**1

StateTextItem Enabled
195 273 215 333
S**0

TYPE DLOG

,309 (36)
Characteristic Equation Coefficient Data
40 56 315 456
Visible NoGoAway
1
0
309

TYPE DITL
,309
23

BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380
Cancel

EditTextItem Enabled
75 20 95 130

EditTextItem Enabled
130 20 150 130

EditTextItem Enabled
130 145 150 255

EditTextItem Enabled
130 270 150 380

EditTextItem Enabled
185 20 205 130

EditTextItem Enabled
185 145 205 255

EditTextItem Enabled
185 270 205 380

EditTextItem Enabled
240 20 260 130

EditTextItem Enabled
240 145 260 255

EditTextItem Enabled
240 270 260 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
50 23 70 83
S**9

StateTextItem Enabled

```
105 23 125 83
S**8

StateTextItem Enabled
105 148 125 208
S**7

StateTextItem Enabled
105 273 125 333
S**6

StateTextItem Enabled
160 23 180 83
S**5

StateTextItem Enabled
160 148 180 208
S**4

StateTextItem Enabled
160 273 180 333
S**3

StateTextItem Enabled
215 23 235 83
S**2

StateTextItem Enabled
215 148 235 208
S**1

StateTextItem Enabled
215 273 235 333
S**0

TYPE DLOG

  ,310 (36)
Characteristic Equation Coefficient Data
40 56 315 456
Visible NoGoAway
1
0
310

TYPE DITL
  ,310
```

25

BtnItem Enabled
15 320 35 380
OK

BtnItem Enabled
50 320 70 380
Cancel

EditTextItem Enabled
75 20 95 130

EditTextItem Enabled
75 145 95 255

EditTextItem Enabled
130 20 150 130

EditTextItem Enabled
130 145 150 255

EditTextItem Enabled
130 270 150 380

EditTextItem Enabled
185 20 205 130

EditTextItem Enabled
185 145 205 255

EditTextItem Enabled
185 270 205 380

EditTextItem Enabled
240 20 260 130

EditTextItem Enabled
240 145 260 255

EditTextItem Enabled
240 270 260 380

StateTextItem Enabled
20 15 40 305
Characteristic Equation Coefficient Data

StateTextItem Enabled
50 23 70 83
S**10

StateTextItem Enabled
50 148 70 208
S**9

StateTextItem Enabled
105 23 125 83
S**8

StateTextItem Enabled
105 148 125 208
S**7

StateTextItem Enabled
105 273 125 333
S**6

StateTextItem Enabled
160 23 180 83
S**5

StateTextItem Enabled
160 148 180 208
S**4

StateTextItem Enabled
160 273 180 333
S**3

StateTextItem Enabled
215 23 235 83
S**2

StateTextItem Enabled
215 148 235 208
S**1

StateTextItem Enabled
215 273 235 333
S**0

TYPE MENU

,1000

```
\14
About MacRootLocus ...
(-

 ,1001
File
EQ Parameter/E
Get Coeff/G
Print Screen  /S
Print Window  /R
Quit/Q

 ,1002
Edit
Undo  /U
(-
Cut  /X
Copy  /C
Paste  /V
Clear

 ,1003
Plot
One Parameter/O
Two Parameter/T

 ,1004
Help
EQ Parameter/A
Get Coeff/F
One Parameter/N
Two Parameter/W
Print Out/P
```

LIST OF REFERENCES

1. Apple Computer, Inc.,*Macintosh System Software User's Guide*, pp. 123–154,1988.

2. Graham, N., *Introduction to Computer Science: A Structured Approach*, 2nd ed., pp. 239 – 249, West Publishing Company, 1982

3. Ralston, A., *A First Course in Numerical Analysis,* pp. 368 – 371, McGraw–Hill Book Co., 1978.

4. Borland International, Inc., *Turbo Pascal Tutor (a Self–study Guide to Turbo Pascal on the Macintosh)*, 1987.

5. Borland International, Inc., *Turbo Pascal for the Mac (User's Guide and Reference Manual)*, 1986.

INITIAL DISTRIBUTION LIST

		NO.Copies
1.	Defence Technical Information Center Cameron Station Arlington, Virgini 22304–6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943–5002	2
3.	Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey CA 93943–5000	1
4.	Professor George J. Thaler, Code 62Tr Naval Postgraduate School Monterey CA 93943–5000	5
5.	Professor Harold A. Titus, Code 67Ts Naval Postgraduate School Monterey CA 93943–5000	1
6.	Professor SeHung Kwak, Code 52Kw Naval Postgraduate School Monterey CA 93943–5000	1
7.	Professor James W. Bamham Assoc. Head. Dept of Mech Engineering & Mechanics College of Engineering Drexel Univ. Philadelphia PA, 19104	1
8.	Major Ko, Sung Hoon 286–38 SooYu 4 Dong DoBong Gu Seoul 132 Seoul Korea	5
9.	Park, Seung Chin 3401 N. Columbus #16j Tucson AZ. 85712	1
10.	Kim Yoo Chang 460 W. Forest ave #903 Detroit Michigan 48201	1

11. Lcdr Kenneth Mac Donald 1
1524 Dolphin Court
Orange Park Florida 32673

12. Lcdr Roy Wood 1
1522 Oak Knoll Road
Virginia Beach, Virginia 23414

13. Lcdr Hwang, Jung Sub 1
SMC1209 Naval Postgraduate School
Monterey, CA 93943

14. Seo Yong Seok 1
SMC1448 Naval Postgraduate School
Monterey, CA 93943

15. Hong–on Kim 1
SMC2665 Naval Postgraduate School
Monterey, CA 93943

16. Kang, Mung Hung 1
SMC1375 Naval Postgraduate School
Monterey, CA 93943

17. Kwon, Hui Man 1
SMC1375 Naval Postgraduate School
Monterey, CA 93943

18. Yang, Young Hyee 1
SMC2440 Naval Postgraduate School
Monterey, CA 93943

Thesis
K716248 Ko
c.1 MacRootLocus, a CAD
design tool for feed-
back control systems.